Universidad Carlos III de Madrid

Escuela Politécnica Superior

Dpto. de INGENIERÍA DE SISTEMAS Y AUTOMÁTICA

Ingeniería Técnica Industrial en

Electrónica Industrial

Proyecto Fin de Carrera

IMPLEMENTACIÓN DE UN SENSOR DE FUERZA PAR EN UN ROBOT ABB IRB 1600 (II)

Alumno: Razvan Bogdan
Tutor: Santiago Martínez de la Casa

Leganés, 2012

ÍNDICE

I. Lista de tablas.

II. Lista de figuras.

III. Lista de códigos.

1. INTRODUCCIÓN.

1.1. Descripción del proyecto.

Este proyecto fin de carrera se basa en la implementación de una aplicación la cual permita la comunicación de un PC, un sensor fuerza par JR3 y un robot ABB IRB 1600.

Mediante SOLIDEDGE, se procede al diseño de las piezas necesarias para la instalación del sensor fuerza-par JR3.

En concreto, la aplicación debe permitir que el usuario pueda aplicar una fuerza a la herramienta situada en el extremo del robot ABB y mediante la lectura e interpretación de dicha fuerza, el robot se mueva desde su posición actual hasta una posición final en la misma dirección y una distancia proporcional a la fuerza aplicada.

Mediante la modificación del software facilitado por JR3 para la captura de datos del sensor, se crea una nueva aplicación la cual incluye una comunicación mediante un socket (TCP/IP) para el envío de un paquete de datos donde se incluyen los valores de fuerzas en los tres ejes cartesianos. Además, se incorpora al software la posibilidad de almacenar los datos extraídos del sensor JR3.

La modificación de dicha aplicación se desarrolla mediante lenguaje C++ en Microsoft Visual Studio 2008.

Una vez que los datos son recibidos por el robot ABB, el procesamiento de estos se programa en Robotstudio.

1.2. Objetivos.

El objetivo principal de este Proyecto Fin de Carrera es el diseño de una aplicación la cual proporcione al robot ABB la comunicación con un PC que le permita recibir los valores de fuerza aplicados en su herramienta para interpretarlos y realizar movimientos en la misma dirección, sentido y desplazándose una distancia proporcional a la fuerza aplicada.

Esta parte II del proyecto se encuentra formado por dos objetivos claramente diferenciados.

1. HARDWARE :

Este apartado comprende todo lo relacionado con la implementación física del sensor de fuerza-par JR3 al brazo robotizado ABB IRB 1600.

El primero de los objetivos a desarrollar es la búsqueda de una ubicación para el sensor y sus conexiones eléctricas que no interfiera y bloque con el movimiento de las articulaciones que el brazo robotizado posee.

Una vez demarcada la ubicación del sensor de fuerza-par JR3 la segunda parte es el diseño y posterior implementación de forma física de un sistema de fijación de un mango metálico al sensor de fuerza-par.

Estas dos piezas a su vez quedaran fijadas al robot ABB IRB 1600.

2. SOFTWARE :

En la parte de programación se procederá a la modificación del programa suministrado por el fabricante del sensor de fuerza-par JR3 para que realice los siguientes subobjetivos:

- Crear comunicación mediante socket (TCP/IP) para el envío del paquete de datos con los valores de las fuerzas en los tres ejes cartesianos.

- Creación de un único paquete de datos con los tres valores de fuerza a enviar al robot para crear así un proceso más rápido y obtener de este modo una comunicación más fluida.

- Implementación mediante software de la opción de almacenamiento de datos extraídos del sensor fuerza-par JR3 para un mejor estudio.

- Modificación de la interfaz original adaptándola mejor a las necesidades del proyecto.

El procesamiento de los datos recibidos por el robot para ejecutar una trayectoria de forma simétrica a la fuerza ejercida por la persona se encuentra explicado en la parte I del proyecto.

1.3. Estructura del proyecto.

El proyecto se divide en 12 capítulos que se describen a continuación.

En el Capítulo 1 se realiza una descripción de los temas a tratar en el Proyecto Fin de Carrera así como una descripción de las partes que lo forman.

El Capítulo 2 realiza una descripción histórica del robot ABB y el sensor JR3 para conocer mejor su funcionamiento.

En el Capítulo 3 se describen y explican todos los elementos físicos del proyecto y se da a conocer el software empleado para el desarrollo de la aplicación. En este capítulo se realiza un breve tutorial con las principales funcionalidades de los entornos utilizados.

El Capitulo 4 explica todo el trabajo de desarrollo de la parte física del proyecto.

En el Capítulo 5 se describe todo el trabajo de desarrollo software. Desde la recepción de datos del sensor, el envió de estos y su posterior procesado por el robot hasta las modificaciones de la interfaz de control para el usuario.

El Capítulo 6 explica en forma de tutorial, como llevar a cabo de modo practico la puesta en marcha del sistema.

El Capítulo 7 es en el cual se interpretan los valores obtenidos y se realizan todas las pruebas necesarias para comprobar la correcta lectura de los valores de fuerza obtenidos por el sensor JR3.

El presupuesto se encuentra en el capítulo 8.

El Capítulo 9 trata las conclusiones obtenidas tras la realización de este proyecto, mientras que en el capítulo 10 se dan a conocer posibles trabajos futuros y mejoras de lo realizado.

En el Capítulo 11 se recoge toda la bibliografía utilizada en el proyecto.

Por último, en el capítulo 12 se encuentran los anexos del proyecto donde se puede ver el código integro utilizado, y los planos de la parte hardware implementada.

2. FUNDAMENTOS DEL PROYECTO.

2.1. Robot manipulador.

2.1.1. Definición de robot industrial

Existen ciertas dificultades a la hora de establecer una definición formal de lo que es un robot industrial [1]. La primera de ellas surge de la diferencia conceptual entre definir esta herramienta industrial en el mercado oriental frente al occidental. Así, mientras que en zonas orientales, un robot industrial es cualquier dispositivo mecánico dotado de articulaciones móviles destinado a la manipulación, el mercado occidental es más restrictivo, exigiendo una mayor complejidad, sobre todo en lo relativo al control. En segundo lugar, y centrándose ya en el concepto occidental, aunque existe una idea común acerca de lo que es un robot industrial, no es fácil ponerse de acuerdo a la hora de determinar una definición formal. Además, la evolución de la robótica ha ido obligando a diferentes actualizaciones de su definición.

La definición más comúnmente aceptada posiblemente sea la de la Asociación de Industrias de Robótica (RIA, *Robotic Industry Association*), según la cual:

"Un robot industrial es un manipulador multifuncional reprogramable, capaz de mover materias, piezas, herramientas, o dispositivos especiales, según trayectorias variables, programadas para realizar tareas diversas"

Esta definición, ligeramente modificada, ha sido adoptada por la Organización Internacional de Estándares (ISO) que define al robot industrial como:

"Manipulador multifuncional reprogramable con varios grados de libertad, capaz de manipular materias, piezas, herramientas o dispositivos especiales según trayectorias variables programadas para realizar tareas diversas"

Se incluye en esta definición la necesidad de que el robot tenga varios grados de libertad. Una definición más completa es la establecida por la Asociación Francesa de Normalización (AFNOR), que define primero el manipulador y, basándose en dicha definición, el robot:

- **Manipulador**: mecanismo formado generalmente por elementos en serie, articulados entre sí, destinado al agarre y desplazamiento de objetos. Es multifuncional y puede ser gobernado directamente por un operador humano o mediante dispositivo lógico.

- **Robot**: manipulador automático servo-controlado, reprogramable, polivalente, capaz de posicionar y orientar piezas, útiles o dispositivos especiales, siguiendo trayectoria variables reprogramables, para la ejecución de tareas variadas. Normalmente tiene la forma de uno o varios brazos terminados en una muñeca. Su unidad de control incluye un dispositivo de memoria y ocasionalmente de percepción del entorno. Normalmente su uso es el de realizar una tarea de manera cíclica, pudiéndose adaptar a otra sin cambios permanentes en su material.

Por último, la Federación Internacional de Robótica (IFR, *International Federation of Robotics*) distingue entre robot industrial de manipulación y otros robots:

"Por robot industrial de manipulación se entiende una máquina de manipulación automática, reprogramable y multifuncional con tres o más ejes que pueden posicionar y orientar materias, piezas, herramientas o dispositivos especiales para la ejecución de trabajos diversos en las diferentes etapas de la producción industrial, ya sea en una posición fija o en movimiento"

En esta definición se debe entender que la reprogramabilidad y la multifunción se consiguen sin modificaciones físicas del robot.

Común en todas las definiciones anteriores es la aceptación del robot industrial como un brazo mecánico con capacidad de manipulación y que incorpora un control más o menos complejo. Un sistema robotizado, en cambio, es un concepto más amplio. Engloba todos aquellos dispositivos que realizan tareas de forma automática en sustitución de un ser humano y que pueden incorporar o no a uno o varios robots, siendo esto último lo más frecuente.

2.1.2. Historia y evolución de los robots.

El concepto de máquinas automatizadas se remonta a la antigüedad, con mitos de seres mecánicos vivientes. Los autómatas, o máquinas semejantes a personas, ya aparecían en los relojes de las iglesias medievales, y los relojeros del siglo XVIII eran famosos por sus ingeniosas criaturas mecánicas [2].

El control por realimentación, el desarrollo de herramientas especializadas y la división del trabajo en tareas más pequeñas que pudieran realizar obreros o máquinas fueron ingredientes esenciales en la automatización de las fábricas en el siglo XVIII. A medida que mejoraba la tecnología se desarrollaron máquinas especializadas para tareas como poner tapones a las botellas o verter caucho líquido en moldes para neumáticos. Sin embargo, ninguna de estas máquinas tenía la versatilidad del brazo humano, y no podía alcanzar objetos alejados y colocarlos en la posición deseada.

En la década de 1890 el científico Nikola Tesla, inventor, entre muchos otros dispositivos, de los motores de inducción, ya construía vehículos controlados a distancia por radio. Tesla fue un visionario que escribió sobre mecanismos inteligentes tan capaces como los humanos.

Las máquinas más próximas a lo que hoy en día se entiende como robots fueron los "teleoperadores", utilizados en la industria nuclear para la manipulación de sustancias radiactivas. Básicamente se trataba de servomecanismos que, mediante sistemas mecánicos, repetían las operaciones que simultáneamente estaba realizando un operador.

Inmediatamente después de la Segunda Guerra Mundial comienzan los primeros trabajos que llevan a los robots industriales. A finales de los 40 se inician programas de investigación en los laboratorios de *Oak Ridge y Argonne National Laboratories* para desarrollar manipuladores mecánicos para elementos radiactivos. Estos manipuladores eran del tipo "maestro-esclavo", diseñados para que reprodujeran fielmente los movimientos de brazos y manos realizados por un operario.

El inventor estadounidense George C. Devol desarrolló en 1954 un *dispositivo de transferencia programada articulada* (según su propia definición); un brazo primitivo que se podía programar para realizar tareas específicas.

En 1958, Devol se unió a Joseph F. Engelberger y, en el garaje de este último, construyeron un robot al que llamaron *Unimate.* Era un dispositivo que utilizaba un computador junto con un manipulador que conformaban una "máquina" que podía ser "enseñada" para la realización de tareas variadas de forma automática. En 1962, el primer *Unimate* fue instalado a modo de prueba en una planta de la *General Motors* para funciones de manipulación de piezas y ensamblaje, con lo que pasó a convertirse en el primer robot *industrial*. Devol y Engelberger fundarían más tarde la primera compañía dedicada expresamente a fabricar robots, *Unimation* (véase Fig 2.1), *Inc.*, abreviación de *Universal Automation.*

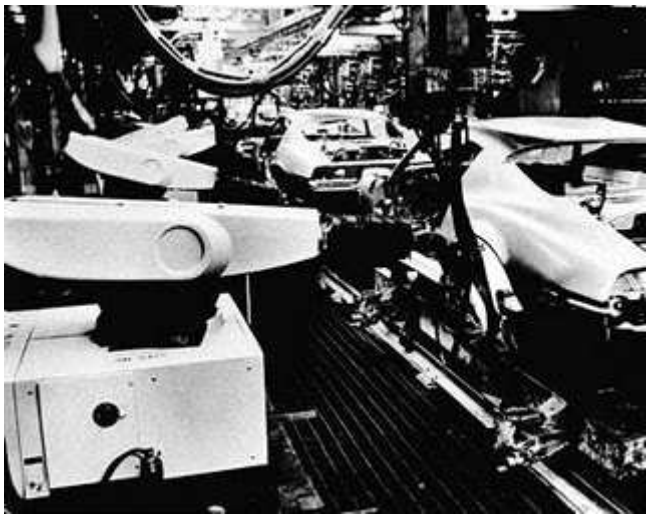


Fig. 2.1. Robot Unimate en línea de ensamble de GM.

Se puede considerar este punto como el inicio de la era de la Robótica tal como la conocemos, mediante la utilización de los robots programados, una nueva y potente herramienta de fabricación.

Durante la década de los 60, un nuevo concepto surge en relación con los anteriores avances. En vistas a una mayor flexibilidad, se hace necesaria la realimentación sensorial. En 1962, H. A. Ernst publica el desarrollo de una mano mecánica controlada por computador con sensores táctiles llamada *MH-1*. Este modelo evolucionó adaptándole una cámara de televisión dentro del proyecto *MAC*. También en 1962, Tomovic y Boni desarrollan una mano con un sensor de presión para la detección del objeto que proporcionaba una señal de realimentación al motor.

En 1963 se introduce el robot comercial *VERSATRAN* por la *American Machine and Foundry Company (AMF)*. En el mismo año se desarrollan otros brazos manipuladores como el *Roehampton* y el *Edinburgh.*

En 1967 y 1968 *Unimation* recibe sus primeros pedidos para instalar varios robots de la serie *Unimate 2000* en las cadenas de montaje de *General Motors*. Al año siguiente los robots ensamblaban todos los coches *Chevrolet Vega* de esta compañía.

En 1968 se publica el desarrollo de un computador con "manos", "ojos" y "oídos" (manipuladores, cámaras de TV y micrófonos) por parte deMcCarthy en el *Stanford Artificial Intelligence Laboratory.* En el mismo año, Pieper estudia el problema cinemático de un manipulador controlado por un computador. También este año, la compañía japonesa *Kawasaki Heavy Industries* negocia con *Unimation* la licencia de sus robots. Este momento marca el inicio de la investigación y difusión de los robots industriales en Japón. En la Fig 2.2 se puede observar un ejemplo de brazo manipulador fabricado en estas fechas.

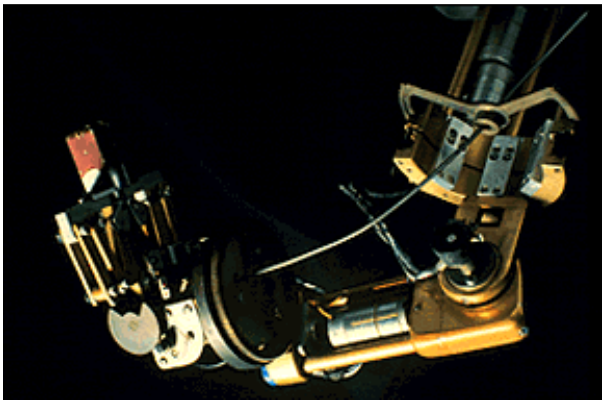


Fig. 2.2. Brazo Stanford (1969).

Las primeras aplicaciones industriales en Europa, aplicaciones de robots industriales en cadenas de fabricación de automóviles, datan de los años 1970 y 1971. En este último año, **Kahn** y **Roth** analizan el comportamiento dinámico y el control de un brazo manipulador.

Durante la década de los 70, la investigación en robótica se centra en gran parte en el uso de sensores externos para su utilización en tareas de manipulación. Es también en estos años cuando se consolida definitivamente la presencia de robots en las cadenas de montaje y plantas industriales en el ámbito mundial.

En 1972 se desarrolló en la universidad de *Nottingham*, Inglaterra, el *SIRCH*, un robot capaz de reconocer y orientar objetos en dos dimensiones. Este mismo año, la empresa japonesa *Kawasaki* instala su primera cadena de montaje automatizada en *Nissan*, Japón, usando robots suministrados por *Unimation, Inc*.

En 1973, **Bolles** y **Paul** utilizan realimentación visual en el brazo *Stanford* para el montaje de bombas de agua de automóvil. También este mismo año, la compañía sueca *ASEA* (futura *ABB*), lanza al mercado su familia de robots *IRB 6* e *IRB 60*, para funciones de perforación de piezas.

También este mismo año, la empresa *Cincinatti Milacron* introduce el T3 (*The Tomorrow Tool*), su primer robot industrial controlado por computador. Este manipulador podía levantar más de 45 kilogramos y seguir objetos móviles en una línea de montaje.

En 1975, **Will** y **Grossman**, en *IBM*, desarrollaron un manipulador controlado por computador con sensores de contacto y fuerza para montajes mecánicos. Este mismo año, el ingeniero mecánico estadounidense **Victor Scheinman**, cuando estudiaba la carrera en la Universidad de*Stanford*, California, desarrolló un manipulador polivalente realmente flexible conocido como *Brazo Manipulador Universal Programable* (*PUMA*, siglas en inglés). El *PUMA* era capaz de mover un objeto y colocarlo en cualquier orientación en un lugar deseado que estuviera a su alcance. El concepto básico multiarticulado del *PUMA* (véase Fig 2.3) es la base de la mayoría de los robots actuales.

Fig. 2.3. Robot PUMA.

En 1976, estudios sobre el control dinámico llevados a cabo en los laboratorios *Draper*, Cambridge, permiten a los robots alinear piezas con movimientos laterales y rotacionales a la vez.

En 1979 Japón introduce el robot *SCARA* (*Selective Compliance Assembly Robot Arm*), y la compañía italiana *DEA* (*Digital Electric Automation*), desarrolla el robot *PRAGMA* para la *General Motors*.

En la década de los 80 se avanza en las técnicas de reconocimiento de voz, detección de objetos móviles y factores de seguridad. También se desarrollan los primeros robots en el campo de la rehabilitación, la seguridad, con fines militares y para la realización de tareas peligrosas. Así por ejemplo, en 1982, el robot *Pedesco*, se usa para limpiar un derrame de combustible en una central nuclear. También se pone un gran énfasis en los campos de visión artificial, sensorización táctil y lenguajes de programación. Gracias a los primeros pasos dados por compañías como *IBM* o *Intelledex Corporation*, que introdujo en 1984 el modelo ligero de ensamblaje 695, basado en el microprocesador *Intel* 8087 y con software *Robot Basic*, una modificación del *Microsoft Basic*, actualmente se tiende al uso de una interfaz (el ordenador) y diversos lenguajes de programación especialmente diseñados, que evitan el "cuello de botella" que se producía con la programación "clásica". Esta puede ser ahora *on-line* u *off-line*, con interfaces gráficas (*user-friendly interfaces*) que facilitan la programación, y un soporte *SW+HW* que tiende a ser cada vez más versátil.

2.1.3. Clasificación de los robots.

La clasificación de los robots industriales se puede hacer de muchos tipos, dependiendo de los aspectos que se deseen reflejar. Una primera clasificación genérica seria esta [3]:

Tipo de Robots	Características
Manipuladores	Son sistemas mecánicos multifuncionales, con un sencillo sistema de control, que permite gobernar el movimiento de sus elementos.
Robots de repetición o aprendizaje	Son manipuladores que se limitan a repetir una secuencia de movimientos, previamente ejecutada por un operador humano, haciendo uso de un controlador manual o un dispositivo auxiliar.
Robots de control por computador	Son manipuladores o sistemas mecánicos multifuncionales, controlados por un computador, que habitualmente suele ser un microordenador. Las grandes ventajas que ofrecen este tipo de robots, hacen que se vayan imponiendo en el mercado rápidamente.
Robos inteligentes	Son similares a los anteriores, pero, además, son capaces de relacionarse con el mundo que les rodea a través de sensores y tomar decisiones en tiempo real.

Tabla 2 1. Clasificación genérica de robots.

Por otra parte, se puede realizar otro tipo de clasificación analizando en este caso la generación a la que pertenece el robot industrial.

Tipo de Robots	Características
Robots de Primera Generación.	- Repite la tarea programada secuencialmente. No toma en cuenta las posibles alteraciones de su entorno. - Los dispositivos actúan como "esclavo" mecánico de un hombre, quien provee mediante su intervención directa el control de los órganos de movimiento. - Esta transmisión tiene lugar mediante servomecanismos actuados por las extremidades superiores del hombre. Casos típicos son la manipulación de materiales radiactivos, obtención de muestras submarinas, etc.
Robots de Segunda Generación.	- Adquiere información limitada de su entorno y actúa en consecuencia. Puede localizar, clasificar (visión) y detectar esfuerzos y adaptar sus movimientos en consecuencia. - El dispositivo actúa automáticamente sin intervención humana frente a posiciones fijas en las que el trabajo ha sido preparado y ubicado. - Ejecuta movimientos repetitivos en el tiempo, que obedecen a lógicas combinatorias, secuenciales, programadores paso a paso, neumáticos o Controladores Lógicos Programables. - Tienen una facilidad para una rápida reprogramación lo que convierte a estos Robots en unidades "versátiles" cuyo campo de aplicación no sólo se encuentra en la manipulación de materiales sino en todo los procesos de manufactura. - Usos: en el estampado en frío y en caliente asistiendo a las máquinas; herramientas para la carga y descarga de piezas; en la inyección de termoplásticos y metales no ferrosos; en los procesos de soldadura a punto y continúa; en tareas de pintado; y reemplazando con ventaja algunas operaciones de máquinas convencionales.
Robots de Tercera Generación.	- Su programación se realiza mediante el empleo de un lenguaje natural. Posee la capacidad para la planificación automática de sus tareas. - Son dispositivos que habiendo sido construidos para alcanzar determinados objetivos serán capaces de elegir la mejor forma de hacerlo teniendo en cuenta el ambiente que los circunda. - Deben poseer algunas condiciones que posibiliten su interacción con el ambiente y los objetos. Las mínimas aptitudes requeridas son: capacidad de reconocer un elemento determinado en el espacio y la capacidad de adoptar propias trayectorias para conseguir el objetivo deseado. - Los métodos de identificación empleados hacen referencia a la imagen óptica por ser esta el lenguaje humano en la observación de los objetos, sin embargo no puede asegurarse que la que es natural para el hombre, constituye la mejor solución para el robot.

Tabla 2 2. Clasificación de robots según generación.

Por último, se describe en la siguiente tabla la clasificación de robots industriales dependiendo de su estructura mecánica y de los tipos de movimientos que este puede realizar.

Tipo	Descripción
Robot cartesiano	Robot donde el brazo está constituido por tres articulaciones prismáticas donde los ejes están dispuestos según un sistema de coordenadas cartesianas.
Robot cilíndrico	Robot donde el brazo está constituido por al menos una articulación rotativa y de una articulación prismática donde los ejes están dispuestos según un sistema de coordenadas cilíndricas.
Robot Esférico (o polar)	Robot cuyos ejes forman un sistema de coordenadas polar.
Robot SCARA	Robot que tiene dos articulaciones rotativas con ejes paralelos, para proporcionar conformidad en un plano dado.
Robot articulado	Robot cuyo brazo tiene al menos articulaciones rotativas
Robot paralelo	Robot donde el brazo (ejes principales) están constituidos por tres articulaciones prismáticas concurrentes

Tabla 2 3. Clasificación de robots según su estructura mecánica.

2.2. Sensores.

2.2.1. Definición de sensor.

Un sensor es un dispositivo que está capacitado para detectar acciones o estímulos externos y responder en consecuencia**.** Estos aparatos pueden transformar las magnitudes físicas o químicas en magnitudes eléctricas.

Por ejemplo: existen sensores que se instalan en los vehículos y que detectan cuando la velocidad de desplazamiento supera la permitida. En esos casos, emiten un sonido que alerta al conductor y a los pasajeros.

Otro tipo de sensor muy habitual es aquel que se instala en la puerta de entrada de las viviendas y reacciona ante e movimiento. Si una persona se acerca al sensor, éste emite una señal y se enciende la iluminación. La utilización de estos sensores está vinculada a la seguridad, ya que evitan que alguien aproveche la oscuridad para ocultarse e ingrese en la casa sin ser advertido.

Los termómetros también son sensores ya que aprovechan la capacidad del mercurio para reaccionar ante la temperatura y, de este modo, permiten detectar si una persona tiene fiebre.

Los sensores, en definitiva, son dispositivos que permiten obtener información del entorno e interactuar con ella.

Cuando se desarrollan computadoras capaces de responder a órdenes de voz, por ejemplo, se las provee de micrófonos que son sensores capaces de captar las ondas sonoras y transformarlas. Si estos sensores están asociados a otros circuitos, la máquina podrá reaccionar al estímulo de acuerdo a lo requerido por el usuario.

2.2.2. Historia de los sensores.

Al igual que los seres humanos apelan a su sistema sensorial (véase Fig 2.3) para obtener información del entorno, las máquinas y los robots requieren de sensores para la interacción con el medio en el que se encuentran [5].

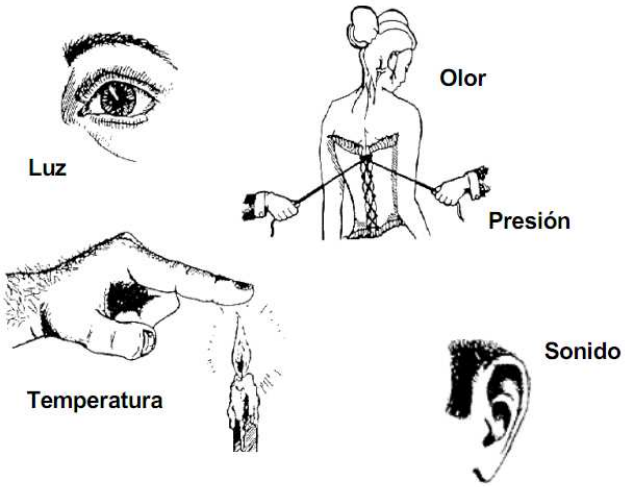


Fig. 2.4. Sistema sensorial humano.

Los sensores existen desde siempre, y nunca mejor dicho, porque el hombre los tiene incluidos en su cuerpo y de diferentes tipos. El hombre experimenta sensaciones como calor o frío, duro o blando, fuerte o flojo, agradable o desagradable, pesado o no. Y poco a poco le ha ido añadiendo adjetivos a estas sensaciones para cuantificarlas como frígido, fresco, tibio, templado, caliente, tórrido. Es decir, que día a día ha ido necesitando el empleo de magnitudes medibles más exactas.

2.2.3. Características de los sensores.

2.2.3.1 Características estáticas de un sensor.

Las características estáticas definen el comportamiento en régimen permanente del sensor:

- Rango: Valores máximos y mínimos para las variables de entrada y salida de un sensor.
- Exactitud: la desviación de la lectura de un sistema de medida respecto a una entrada conocida. El mayor error esperado entre las señales medida e ideal.
- Repetitividad: la capacidad de reproducir una lectura con una precisión dada.

- Reproducibilidad: tiene el mismo sentido que la repetitividad excepto que se utiliza cuando se toman medidas distintas bajo condiciones diferentes.

- Resolución: la cantidad de medida más pequeña que se pueda detectar.

- Error: es la diferencia entre el valor medido y el valor real.

- No linealidades: la desviación de la medida de su valor real, supuesto que la respuesta del sensor es lineal. No-linealidades típicas: saturación, zona muerta e histéresis.

- Sensibilidad: es la razón de cambio de la salida frente a cambios en la entrada: $s = \partial V / \partial x$.

- Excitación: es la cantidad de corriente o voltaje requerida para el funcionamiento del sensor.

- Estabilidad: es una medida de la posibilidad de un sensor de mostrar la misma salida en un rango en que la entrada permanece constante.

- Ruido: se trata de la señal no deseada que se mezcla con la útil.

2.2.3.2 Características dinámicas de un sensor.

Las características dinámicas definen el comportamiento en régimen transitorio del sensor. En la Fig 2.4 se observan las distintas características dinámicas de un sensor.

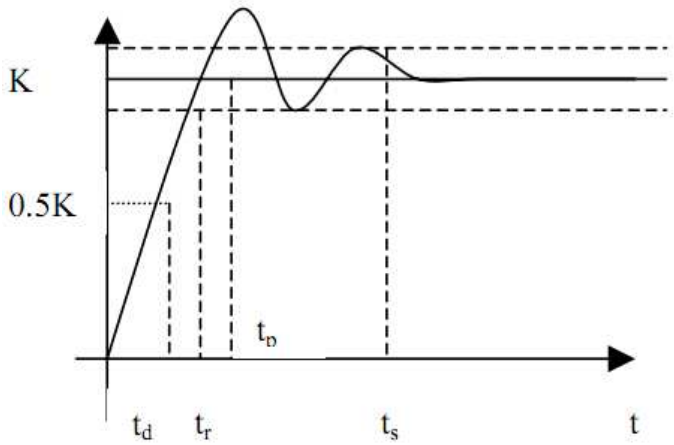


Fig. 2.5. Características dinámicas de un sensor.

Las características dinámicas más importantes son:

- Tiempo de retardo: td, es el tiempo que tarda la salida del sensor en alcanzar el 50% de su valor final.

- Tiempo de subida: tr, es el tiempo que tarda la salida del sensor hasta alcanzar su valor final. => velocidad del sensor, es decir, lo rápido que responde ante una entrada.

- Tiempo de pico: tp, es el tiempo que tarda la salida del sensor en alcanzar el pico máximo de su sobreoscilación.

- Pico de sobreoscilación: Mp, expresa cuanto se eleva la evolución temporal de la salida del sensor respecto de su valor final.

- Tiempo de establecimiento: ts, el tiempo que tarda la salida del sensor en entrar en la banda del 5% alrededor del valor final y ya no vuelve a salir de ella

2.2.4. Clasificación de los sensores utilizados en robots.

Para conseguir que el robot realice su tarea con la adecuada precisión es preciso que tenga conocimiento tanto de su propio estado como del estado de su entorno. Dependiendo de su situación, se pueden diferenciar dos tipos de sensores [6]:

- Sensores internos: sensores integrados en la propia estructura mecánica del robot, que dan información del estado del robot: fundamentalmente de la posición, velocidad y aceleración de las articulaciones.

- Sensores externos: dan información del entorno del robot: alcance, proximidad, contacto, fuerza, etc. Se utilizan para guiado de robots, para identificación y manipulación de objetos.

2.2.4.1 Sensores internos.

Permiten a la unidad de control del robot obtener datos del estado de su estructura mecánica como posición, velocidad y aceleración.

2.2.4.1.1 Sensores de posición.

Encargados de indicar a la unidad de control la posición del robot. Se pueden definir dos tipos de sensores de posición principalmente ,analógicos y digitales:

- Analógicos. Es aquel que puede entregar una salida variable dentro de un determinado rango (Vease Fig 2.6).

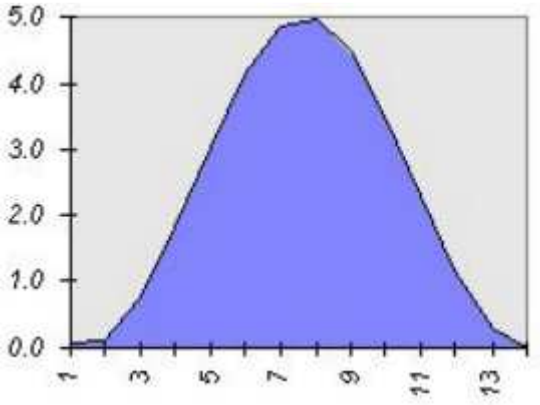


Fig. 2.6. Señal analógica.

Algunos ejemplos de sensores de posición analógicos son:

i. Potenciómetros: Se usan para la determinación de desplazamientos lineales o angulares. Eléctricamente se cumple la relación $\frac{V_{BC}}{V_{AC}} = \frac{R_{BC}}{R_{AC}}$ (véase Fig 2.7).

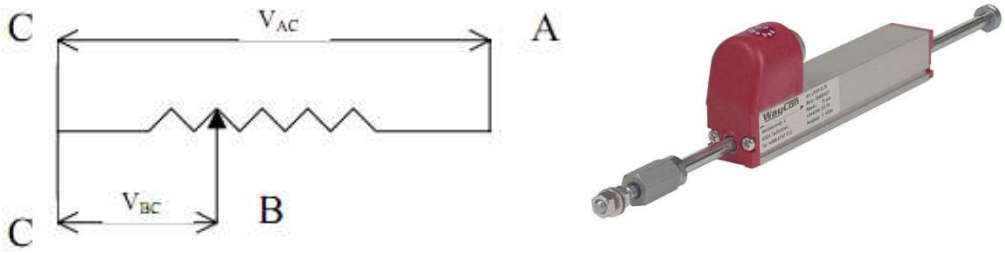


Fig. 2.7. Potenciómetro.

Este potencial puede medirse y disponer de un sistema de calibrado de manera que por cada potencial se obtenga proporcionalmente una distancia de desplazamiento.

Resolver. Los resolvers parecen pequeños motores pero son, esencialmente, transformadores rotativos diseñados de tal modo que su coeficiente de acoplamiento entre el rotor y el estator varía según sea la posición angular del eje (véase Fig 2.8)

Fig. 2.8. Resolvers.

En su diseño más simple, el resolver consta de dos bobinados en el estator, que detectan una señal senoidal de inducción, emitida por un tercer bobinado desde el rotor; una de las bobinas detectoras corresponde al seno y la otra al coseno (están ubicadas en posiciones separadas, obviamente, por un ángulo de 90°). La bobina excitadora del rotor es alimentada por una señal de corriente alterna senoidal que le llega a través de anillos de metal (contactos) ubicados sobre el eje, y escobillas. Este diseño tiene el inconveniente de que el mecanismo de escobillas sufre un desgaste continuo, lo que hace posible, pasado cierto tiempo de uso, que desde ahí se ingrese ruido en la señal. Tanto el rotor como el estator están construidos con un núcleo de hierro laminado.

Existe una solución de resolver sin escobillas, que está implementada por un transformador rotativo, con el primario sobre el estator y el secundario en el rotor.

ii. LVDT. El transformador diferencial de variación lineal es un tipo de transformador eléctrico utilizado para medir desplazamientos lineales. El transformador posee tres bobinas dispuestas extremo con extremo alrededor de un tubo. La bobina central es el devanado primario y las externas son los secundarios. Un centro ferromagnético de forma cilíndrica, sujeto al objeto cuya posición desea ser medida, se desliza con respecto al eje del tubo (véase Fig 2.9).

Fig. 2.9. LVDT.

Cuando una corriente alterna circula a través del primario, causa un voltaje que es inducido a cada secundario proporcionalmente a la inductancia mutua con el primario. La frecuencia del oscilador que causa la corriente alterna está en el rango de 1 a 10 kHz.

A medida que el núcleo se mueve, la inductancia mutua cambia, causando que el voltaje inducido en el secundario cambie. Las bobinas están conectadas en serie pero invertidas, así que el voltaje de salida es la diferencia (por eso es "diferencial") entre los dos voltajes secundarios. Cuando el núcleo está en su posición central, se encuentra equidistante a los dos secundarios, los voltajes inducidos son iguales pero de signo opuesto, así que el voltaje de salida es cero.

Cuando el núcleo es desplazado en una dirección, el voltaje en una bobina aumenta mientras que en la otra disminuye, causando que el voltaje de salida también aumente desde cero hasta su máximo. Este voltaje tiene la misma fase que el voltaje del primario. La magnitud del voltaje de salida es proporcional a la distancia en que fue desplazado el núcleo (hasta cierto límite), por eso el dispositivo es descrito como "lineal". La fase del voltaje indica la dirección del desplazamiento.

Debido a que el núcleo deslizante no toca el interior del tubo, puede moverse prácticamente sin fricción, haciendo del LVDT un dispositivo muy fiable. La ausencia de contactos deslizantes o rotatorios permite que el LVDT sea completamente sellado.

Los LVDT son usados para la realimentación de posición en servomecanismos y para la medición automática en herramientas y muchos otros usos industriales y científicos.

iii. Sincro. Es un transductor de posición angular de tipo electromagnético. Su principio de funcionamiento puede resumirse diciendo que se trata de un

transformador, en el que uno de los devanados es rotativo. La configuración más habitual se basa en situar el primario en el rotor y monofásico y, por otra parte, el secundario en el estator y trifásico.

Cuando se aplica una tensión senoidal al devando primario, se recogen en los devanados secundarios de cada una de las fases tres tensiones, cuya amplitud y fase con respecto a la tensión del primario dependen de la posición angular del rotor. En la Fig 2.10 se representa un ejemplo de transductor sincro.

Fig. 2.10. Sincro.

iv. Inductosyn. Es un transductor electromagnético utilizado para la medida de desplazamientos lineales, con precisión del orden de micras. Se emplea en máquinas medidoras de coordenada y máquinas herramientas de control numérico. El transductor consta de dos partes acopladas magnéticamente, una denominada escala fija y situada paralela al eje de desplazamiento y otra solapada a la anterior deslizante y solidaria a la parte móvil (véase Fig 2.11).

Fig. 2.11. Inductosyn.

La parte fija lleva grabado un circuito impreso con pistas en forma de onda rectangular con un paso p. La parte móvil tiene dos más pequeños, encarados con los de la escala, y desfasados entre si un número entero de pasos más ¼ de paso (análogamente a lo visto para encoders incrementales). Si se excita la parte fija con una señal alterna en cada uno de los circuitos de la parte deslizante se recoge una tensión que es función del desplazamiento lineal y el paso de onda de la escala. La amplitud varía entre un máximo y un mínimo según las que los circuitos fijo y móvil estén enfrentados o decalados ½ de paso. La medida se realiza

sumando el número de ciclos de señal de salida completos, más la variación dentro de un ciclo. La indeterminación del sentido se resuelve comparando la fase de los dos captadores.

- Digitales. Un sensor digital es aquel que entrega una salida del tipo discreta (véase Fig 2.12). Es decir, que el sensor posee una salida que varía dentro de un determinado rango de valores, pero a diferencia de los sensores analógicos, esta señal varía de a pequeños pasos pre-establecidos.

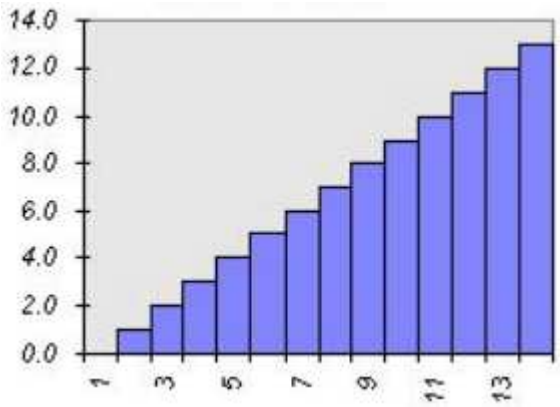


Fig. 2.12. Señal digital.

Algunos ejemplos de sensores de posición digitales son:

i. Encoders: Constan de un disco transparente con una serie de marcas opacas colocadas radialmente y equidistantes entre sí, de un sistema de iluminación y de un elemento fotorreceptor. En la Fig 2.13 se pueden ver las partes y un ejemplo se sensor encoder.

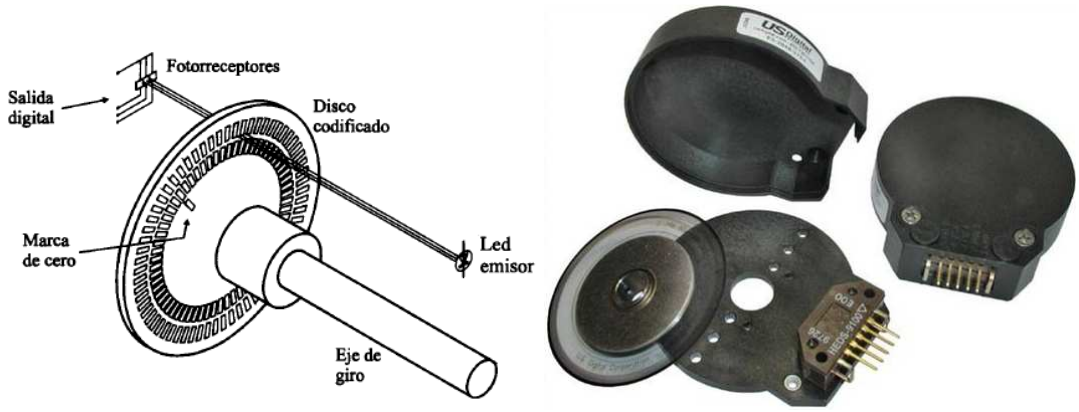


Fig. 2.13. Encoder.

El eje cuya posición se quiere medir va acoplado al disco, a medida que el eje gira se van generando pulsos en el receptor cada vez que la luz atraviese las marcas, llevando una cuenta de estos pulsos es posible conocer la posición del eje.

La resolución depende del número de marcas que se pueden poner físicamente en el disco.

El funcionamiento de un encoder absoluto es similar, pero el disco se divide en un número de sectores (potencia de 2), codificándose cada uno de ellos con un código binario (código Gray), con zonas transparentes y opacas.

2.2.4.1.2 Sensores de velocidad.

La captación de la velocidad se hace necesaria para mejorar el comportamiento dinámico de los actuadores del robot. La información de la velocidad de movimiento de cada actuador se realimenta normalmente a un bucle de control analógico implementado en el propio accionador del elemento motor. No obstante, en las ocasiones en las que el sistema de control del robot lo exija, la velocidad de giro de cada actuador es llevada hasta la unidad de control del robot.

Normalmente, y puesto que el bucle de control de velocidad es analógico, el sensor usado es una **tacogenerador** (véase Fig 2.13) que proporciona una tensión proporcional a la velocidad de giro de su eje. Su funcionamiento se basa en convertir la energía del eje en cuestión en energía eléctrica, proporcional a la rotacional y que puede ser fácilmente medida. Para generar la corriente a partir del giro se acopla al motor o eje que se va a medir, una espira situada dentro de un campo magnético fijo (creado por los dos imanes). Al girar el motor, la espira girará en el interior del campo magnético, lo que provocará una corriente eléctrica.

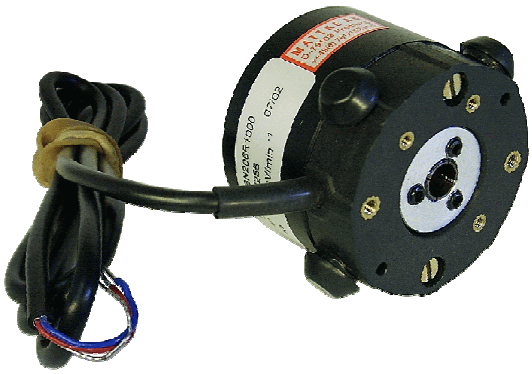


Fig. 2.14. Tacogenerador.

2.2.4.1.3 Sensores de aceleración.

Los sensores de aceleración o acelerómetros son elementos que transforman la magnitud física de aceleración en otra magnitud eléctrica que será la que se emplea en los equipo de automatización o adquisición estándar. Los rangos de medida van desde las decimas de g, hasta los miles de g.

Para cubrir los diferentes rangos de medida, precisión y protección, existen una gran variedad de acelerómetros, fabricados con diferentes tecnologías, que permiten cubrir todas las necesidades. En la Fig 2.15 se muestran unos ejemplos de acelerómetros.

Fig. 2.15. Acelerometros.

2.2.4.2 Sensores externos.

El empleo de mecanismos de detección exteriores permite a un robot interaccionar con su ambiente de una manera flexible. Esto contrasta con el funcionamiento preprogramado en el que a un robot se le enseña a realizar tareas repetitivas mediante una serie de funciones preprogramadas. Aunque esto está bastante lejos de la forma más predominante de funcionamiento de los robots industriales actuales, la utilización de la tecnología de detección para proporcionar a las máquinas un mayor grado de inteligencia en relación con su ambiente es, en realidad, un tema activo de investigación y desarrollo en el campo de la robótica.

Un robot que puede ver y sentir es más fácil de entrenar en la ejecución de las tareas complejas mientras que, al mismo tiempo, exige mecanismos de control menos estrictos que las máquinas preprogramadas. Un sistema sensible y susceptible de entrenamiento es también adaptable a una gama mucho más amplia de tareas, con lo que se consigue un grado de universalidad que se traduce, a la larga, en más bajos costes de producción y mantenimiento. La función de los sensores del robot puede dividirse en dos categorías principales: estado interno y estado externo. Los sensores de estado externo operan con la detección de variables tales como el alcance, la proximidad y el contacto. La detección externa, se utiliza para el guiado del robot, así como para la manipulación e identificación de objetos.

Los sensores de estado externo pueden clasificarse también como sensores de contacto y no contacto. Como su nombre lo indica, la primera clase de sensores responde al contacto físico, tal como el tacto, deslizamiento y torsión. Los sensores de no contacto se basan en la respuesta de un detector a las variaciones en la radiación electromagnética o acústica. Los ejemplos más destacados de los sensores de no contacto miden el alcance, la proximidad y las propiedades visuales de un objeto.

Es de interés destacar que la detección de alcance y la visión suelen proporcionar una información de guiado aproximado para un manipulador, mientras que la proximidad y el tacto están asociados con fases terminales de agarre del objeto. Los sensores de fuerza y torsión se utilizan como dispositivos de retroalimentación para controlar la manipulación de un objeto una vez que haya agarrado.

2.2.4.2.1 Detección de alcance.

Un sensor de alcance mide la distancia desde un punto de referencia (que suele estar en el propio sensor) hasta objetos en el campo de operación del sensor. Los seres humanos estiman la distancia por medio de un procesamiento visual estereográfico. Los sensores de alcance se utilizan para la navegación de robots y para evitar obstáculos, para aplicaciones más detalladas en las que se desean las características de localización y forma en general de objetos en el espacio de trabajo de un robot [7].

- Triangulación.

Uno de los métodos más sencillos para medir alcance es mediante técnicas de triangulación. Su funcionamiento consiste en que un objeto se ilumina por un estrecho haz de luz, que barre toda la superficie. El movimiento de barrido está en el plano definido por la línea desde el objeto hasta el detector y por la línea desde el detector hasta la fuente. Si el detector se enfoca sobre una pequeña parte de la superficie, entonces, cuando el detector vea la mancha luminosa, su distancia a la parte iluminada de la superficie puede calcularse a partir de la geometría de la figura, puesto que se conocen el ángulo de la fuente con la línea de base y la distancia entre la fuente y el detector.

El método anterior proporciona una medida puntual. Si la disposición de fuente-detector se desplaza en un plano fijo (hacia arriba y abajo y en sentido lateral en un plano perpendicular al papel y que contenga la línea de la base en la figura), será posible obtener una serie de puntos cuyas distancias desde el detector serán conocidas. Estas distancias se transforman con facilidad en coordenadas tridimensionales manteniendo un registro de la localización y orientación del detector a medida que se exploran los objetos.

- Método de iluminación estructural.

Este método consiste en proyectar una configuración de luz sobre un conjunto de objetos y en utilizar la distorsión de la sufrida para calcular el alcance.

Una de las configuraciones de luz de mayor difusión actual es una lámina de luz generada a través de una lente cilíndrica o de una hendidura estrecha. La intersección de la lámina de luz con objetos, en el espacio de trabajo, proporciona una franja de luz que se observa a través de una cámara de televisión desplazada en una cierta distancia desde la fuente de luz. La configuración de franjas se analiza con facilidad por una computadora para tener información del alcance. Por ejemplo una inflexión indica un cambio de superficie y una rotura corresponde a una separación entre superficies. Los valores de alcances específicos se calculan calibrando primero el sistema.

En una de las disposiciones más simples, la fuente de luz es perpendicular a la línea que une el origen de dicha lámina y el centro de la lente de la cámara. Al plano vertical que contiene esta línea le llamaremos plano de referencia. Es evidente que el plano de referencia es perpendicular a la lámina de luz y cualquier superficie de plano vertical que corte producirá una franja vertical de luz, en la que cada punto tendrá la misma distancia perpendicular al plano de referencia. El objetivo de la disposición es situar la cámara de modo que cada una de dichas franjas verticales aparezca también vertical en el plano de la imagen. De esta manera, cada punto a lo largo de la misma columna de la imagen será reconocido como que tiene la misma distancia al plano de referencia.

- Telémetro de tiempo de vuelo.

En esta sección se examinan tres métodos para determinar las distancias basadas en el concepto de tiempo de vuelo. Dos de los métodos utilizan un láser, mientras que el tercero está basado en la ultrasónica.

Un método para utilizar un láser para determinar la distancia consistente en medir el tiempo que tarda un pulso de luz emitido para retornar de forma coaxial (es decir, a lo largo de la misma trayectoria) desde una superficie reflectora. La distancia a la superficie viene dada por la simple relación D = c T /2, en donde T es el tiempo de transito del pulso y c es la velocidad de la luz. Es de interés destacar que, puesto que la luz se desplaza a una velocidad aproximada de 1 pie/ns, la instrumentación electrónica de apoyo debe ser capaz de una resolución de tiempo de 50 PS para poder conseguir una exactitud de ± ¼ pulgada en distancia.

Un sistema de láser pulsado descrito por Javis produce una disposición bidimensional con valores proporcionales a la distancia: la exploración bidimensional con valores proporcionales a la distancia. La exploración bidimensional se realiza desviando la luz láser a través de un espejo giratorio. El margen de trabajo de este dispositivo es del orden de magnitud de 1 a 4 metros, con una exactitud de ± 0.25cm. La luz detectada se visualiza como una imagen en la que la intensidad en cada punto es proporcional a la distancia entre el sensor y la superficie reflectora en ese punto (más oscura cuanto más próxima esta). Las zonas brillantes alrededor de los contornos de los objetos representan la discontinuidad en el alcance, determinada mediante un postprocesamiento en una computadora.

Una alternativa a la luz pulsada es utilizar un láser de haz continuo y medir el retardo (es decir, el desplazamiento de fase) entre los haces saliente y de retorno. Supóngase que un haz de luz de láser de longitud de onda l está divido en dos haces. Uno de ellos denominado haz de referencia se desplaza una distancia L de un dispositivo de medición de fase y el otro se desplaza a una distancia D de una superficie reflectora. Puesto que la longitud de onda de la luz láser es pequeña (por ejemplo, 632.8 nm para un láser de helio-neón), el método no resulta práctico para las aplicaciones robóticas. Una solución simple a este problema es modular la amplitud de la luz de láser utilizando una forma de onda de longitud de onda mucho mayor (por ejemplo, una onda sinusoidal moduladora de frecuencia f = 10Mhz tiene una longitud de 30 metros). Pero la señal de referencia es ahora la función modulante. La señal de láser modulada se envía al banco y el haz de retorno de la señal moduladora, que luego se compara con la de referencia para determinar el desplazamiento de fase.

Una ventaja importante en la técnica de la luz continua frente a la luz pulsada es que la primera proporciona información de la intensidad y del alcance. Sin embargo, los sistemas continuos exigen una potencia considerablemente mayor. Las incertidumbres en las mediciones de la distancia obtenidas por una u otra técnica exigen promediar la señal de retorno para reducir el error.

Un telémetro ultrasónico es otro exponente importante del concepto del tiempo de vuelo. La idea básica es la misma que se utiliza con un láser pulsado. Una señal ultrasónica se transmite durante un corto período de tiempo y, puesto que la velocidad de sonido se conoce para un medio de propagación especificado, un simple cálculo, que implica el intervalo de tiempo entre el impulso saliente y el eco de retorno como proporciona una estimación de la distancia a la superficie reflectora. Se utilizan principalmente para navegación y para evitar obstáculos.

2.2.4.2.2 Detección de proximidad.

Los sensores de proximidad suelen tener una salida binaria que indica la presencia de un objeto dentro de un intervalo de distancia especificado. En condiciones normales, los sensores de proximidad se utilizan en robótica para un trabajo en campo cercano en relación a agarrar o evitar un objeto.

- **Sensores inductivos.**

Los sensores basados en un cambio de inductancia debido a la presencia de un objeto metálico están entre los sensores de proximidad industriales de más frecuente uso.

El principio de funcionamiento de estos sensores (véase Fig 2.16) consiste fundamentalmente en una bobina arrollada, situada junto a un imán permanente empaquetado en un receptáculo simple y robusto. El efecto de llevar el sensor a la proximidad de un material ferromagnético produce un cambio en la posición de las líneas de flujo del imán permanente según se indica en la figura. En condiciones estáticas no hay ningún movimiento en las líneas de flujo y, por consiguiente, no se induce ninguna corriente en la bobina. Sin embargo, cuando un objeto ferromagnético penetra en el campo del imán o lo abandona, el cambio resultante en las líneas de flujo induce un impulso de corriente, cuya amplitud y forma son proporcionales a la velocidad de cambio de flujo.

Fig. 2.16. Sensor inductivo.

La forma de onda de la tensión, observada a la salida de la bobina, proporciona un medio efectivo para la detección de proximidad. La tensión medida a través de la bobina varía como una función de la velocidad a la que un material ferromagnético se introduce en el campo del imán. La polaridad de la tensión, fuera del sensor, depende de que el objeto este penetrando en el campo abandonándolo.

Existe una relación entre la amplitud de la tensión y la distancia sensor-objeto. La sensibilidad cae rápidamente al aumentar la distancia, y el sensor sólo es eficaz para fracciones de un milímetro.

Puesto que el sensor requiere movimiento para generar una forma de onda de salida, un método para producir una señal binaria es integrar esta forma de onda. La salida binaria se mantiene a nivel bajo en tanto que le valor integral permanezca por debajo de un umbral especificado, y luego se conmuta a nivel alto (indicando la proximidad de un objeto) cuando se supera el umbral.

- **Sensores de efecto Hall.**

El efecto Hall relaciona la tensión entre dos puntos de un material conductor o semiconductor con un campo magnético a través del material. Cuando se utilizan por sí mismos, los sensores de efecto Hall (véase Fig 2.17) sólo pueden detectar objetos magnetizados. Sin embargo, cuando se emplean en conjunción con un imán permanente son capaces de detectar todos los materiales ferromagnéticos. Cuando se utilizan de dicha manera, un dispositivo de efecto Hall detecta un campo magnético intenso en ausencia de un material ferromagnético en el campo cercano.

Fig. 2.17. Sensor efecto Hall.

Cuando dicho material se lleva a la proximidad del dispositivo, el campo magnético se debilita en el sensor debido a la curvatura de las líneas del campo a través del material.

Los sensores de efecto Hall están basados en el principio de una fuerza de Lorentz que actúa sobre una partícula cargada que se desplaza a través de un campo magnético. Esta fuerza actúa sobre un eje perpendicular al plano establecido por la dirección de movimiento de la partícula cargada y la dirección del campo.

- **Sensores capacitivos.**

A diferencia con los sensores inductivos y de efecto Hall que detectan solamente materiales ferromagnéticos, los sensores capacitivos son potencialmente capaces (con diversos grados de sensibilidad) de detectar todos los materiales sólidos y líquidos. Como su nombre indica, estos sensores están basados en la detección de un cambio en la capacidad inducido por una superficie que se lleva cerca del elemento sensor.

El elemento sensor es un condensador constituido por un electrodo sensible y un electrodo de referencia. Estos electrodos pueden ser, por ejemplo, un disco y un anillo metálicos separados por un material dieléctrico. Una cavidad de aire seco se suele colocar detrás del elemento capacitivo para proporcionar aislamiento. El resto del sensor está constituido por circuitos electrónicos que pueden incluirse como una parte integral de la unidad, en cuyo caso suelen estar embebidos en una resina para proporcionar soporte mecánico y sellado. En la Fig 2.18 se puede observar un ejemplo de sensor capacitivo.

Fig. 2.18. Sensor capacitivo.

- **Sensores ultrasónicos.**

La respuesta de todos los sensores de proximidad hasta ahora examinados depende, en gran medida, del material objeto de la detección. Esta dependencia puede reducirse mucho utilizando sensores ultrasónicos (véase Fig 2.19).

Fig. 2.19. Sensor ultrasónico.

En la estructura de un transductor ultrasónico típico utilizado para detección de proximidad el elemento básico es un transductor electroacústico, frecuentemente del tipo cerámico piezoeléctrico. La capa de resina protege al transductor contra la humedad, polvo y otros factores ambientales y también actúa como un adaptador de impedancia acústica. Puesto que el mismo transductor se suele utilizar para la transmisión y la recepción, un amortiguamiento rápido de la energía acústica es necesario para detectar objetos a pequeña distancia. Esta operación se realiza proporcionando absorbedores acústicos y desacoplando el transductor de su receptáculo. Este último está diseñado de modo que produzca un haz acústico estrecho para una eficaz transferencia de energía y una mejor direccionalidad de la señal.

- **Sensores de proximidad ópticos.**

Los sensores de proximidad ópticos son similares a los sensores ultrasónicos en el sentido de que detectan la proximidad de un objeto por su influencia sobre una onda propagadora que se desplaza desde un transmisor hasta un receptor. Este sensor está constituido por un diodo emisor de luz de estado sólido (LED), que actúa como un transmisor de luz infrarroja y un fotodiodo de estado sólido que actúa como el receptor. En la Fig 2.20 se puede ver un ejemplo de sensor de proximidad óptico.

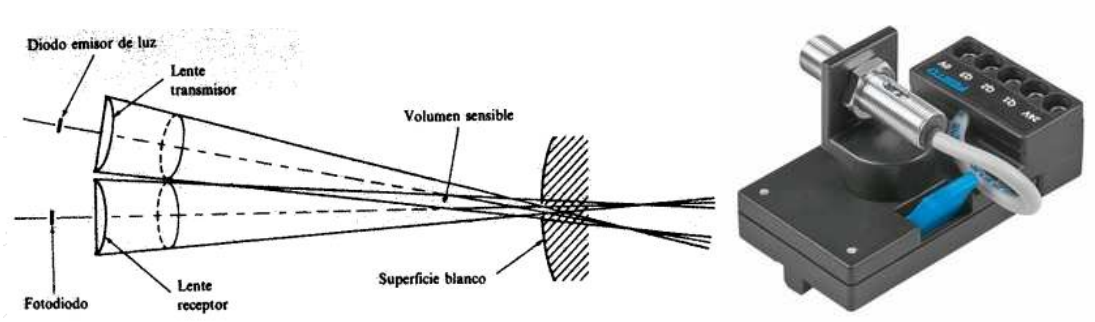


Fig. 2.20. Sensor de proximidad óptico.

Los conos de luz formados enfocando la fuente y el detector en el mismo plano se intersecan en un volumen largo en forma de lápiz. Este volumen define el campo de operación del sensor, puesto que una superficie reflectora que interseca el volumen se ilumina por la fuente y es vista simultáneamente por el receptor.

2.2.4.2.3 Sensores de contacto.

Estos sensores se utilizan en robótica para obtener información asociada con el contacto entre una mano manipuladora y objetos en el espacio de trabajo. Cualquier información puede utilizarse, por ejemplo, para la localización y el reconocimiento del objeto, así como para controlar la fuerza ejercida por un manipulador sobre un objeto dado. Los sensores de contacto pueden subdividirse en dos categorías principales:

- **Sensores binarios.**

Los sensores binarios son dispositivos de contacto tales como micro interruptores. En la disposición más simple, un conmutador está situado en la superficie interior de cada dedo de una mano de manipulación. Este tipo de detección es de utilidad para determinar si una pieza está presente entre los dedos. Desplazando la mano sobre un objeto y estableciendo secuencialmente contacto con la superficie, también es posible centrar la mano sobre el objeto para su agarre y manipulación.

Sensores de contacto binarios múltiples pueden emplearse, en la superficie interior de cada dedo, para proporcionar información táctil. Además, suelen estar montados en las superficies exteriores de una mano de manipulación para proporcionar señales de control de utilidad para guiar la mano a través de todo el espacio de trabajo. Este último modo de detección por contacto es análogo al que los seres humanos sienten cuando se desplazan a través de un recinto completamente oscuro.

- **Sensores analógicos.**

Un sensor de contacto analógico es un dispositivo cuya salida es proporcional a una fuerza local. El más simple de estos dispositivos está constituido por una varilla accionada por resorte que esta mecánicamente enlazada con un eje giratorio, de tal manera que el desplazamiento de la varilla debido a una fuerza lateral da lugar a una rotación proporcional del eje. La rotación se mide luego, de manera continua, utilizando un potenciómetro, o de forma digital, con el uso de una rueda de código. El crecimiento de la constante del resorte proporciona la fuerza que corresponde a un desplazamiento dado.

2.2.4.2.4 Detección de fuerza y torsión.

Los sensores de fuerza y de torsión se utilizan principalmente para medir las fuerzas de reacción desarrolladas en la superficie de separación entre conjuntos mecánicos. Los métodos principales para realizar esta operación son los de detección de:

- **Articulación.**

Un sensor de articulación mide los componentes cartesianos de la fuerza y de la torsión que actúan sobre una articulación de robot y la suma de forma vectorial. Para una articulación impulsada por un motor de corriente continua, la detección se realiza simplemente midiendo la corriente del inducido.

- **Muñeca.**

Los sensores de muñeca están montados entre la extremidad de un brazo del robot y el actuador final. Están constituidos por galgas de deformaciones que miden la desviación de la estructura mecánica debida a fuerzas exteriores. Los sensores de muñeca son pequeños, sensibles, de poco peso y de un diseño relativamente compacto, del orden de 10 cm de diámetro total y de 3cm de espesor. Para poder reducir la histéresis y aumentar la exactitud en la medida, el hardware se suele construir a partir de una pieza mecánica maciza, que suele ser de aluminio. Por ejemplo, el sensor mostrado en la Fig 2.21 utiliza ocho pares de galgas de deformaciones de semiconductores montadas en cuatro barras de deflexión (una galga a cada lado de una barra de deflexión).

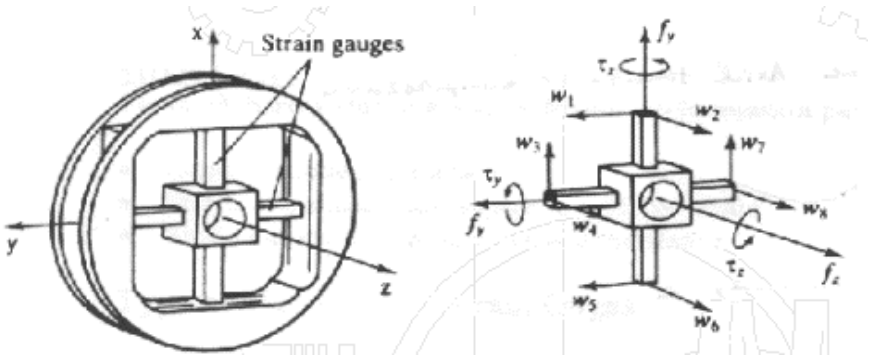


Fig. 2.21. Representación de fuerzas en la muñeca.

Las galgas en los extremos abiertos opuestos de las barras de deflexión están cableadas, de manera diferencial, a un circuito potenciométrico, cuya tensión de salida es proporcional a la componente de la fuerza normal al plano de la galga de deformación.

La conexión diferencial de las galgas de deformación proporciona una compensación automática de las variaciones en la temperatura. Sin embargo, se trata solamente de una compensación de primer orden aproximada. Puesto que los ocho pares de galgas de deformación están orientados en sentido normal a los ejes x, y, z del sistema de referencia, las tres componentes del momento M pueden determinarse sumando y restando adecuadamente las tensiones de salida, respectivamente.

La mayoría de los sensores de fuerza de muñeca funcionan como transductores para transformar las fuerzas y los momentos ejercidos en la mano en desviaciones o desplazamientos medibles en la muñeca generados por el sensor de fuerza no afecten a la exactitud del posicionamiento del manipulador.

Por consiguiente, las especificaciones del rendimiento pueden resumirse como siguen:

- *Alta rigidez.* La frecuencia natural de un dispositivo mecánico está relaciona con su rigidez; así, la alta rigidez asegura que las fuerzas perturbadoras se amortigüen rápidamente para permitir lecturas exactas durante cortos intervalos de tiempo. Además, reduce la magnitud de las desviaciones de una fuerza / momento aplicado, lo que puede añadirse al error de posicionamiento de la mano.

- *Diseño compacto.* Este diseño asegura que el dispositivo no restrinja el movimiento del manipulador en un área de trabajo con poco espacio libre. Con el sensor de fuerza compacto, es importante colocar el sensor lo más próximo posible a la herramienta para reducir el error de posicionamiento como consecuencia del giro de la mano en pequeños ángulos. Además, es deseable medir una fuerza / momento de la mano lo más grande posible; así, al hacer mínima la distancia entre la mano y el sensor, se reduce el brazo de la palanca para las fuerzas aplicadas en la mano.

- *Linealidad.* Una buena linealidad entre la respuesta de los elementos detectores de la fuerza y las fuerzas / momentos aplicados permite resolver la fuerza y los momentos mediante simples operaciones matriciales.

- *Baja histéresis y rozamiento interno.* El rozamiento interno reduce la sensibilidad de los elementos detectores de la fuerza porque las fuerzas tienen que superar este rozamiento, o fricción, antes de que pueda obtenerse una desviación medible.

Produce también efectos de histéresis que no restablecen los dispositivos de medida de la posición a sus lecturas originales.

En la Fig 2.22 se puede observar el sensor de fuerza par como el utilizado en la aplicación desarrollada en el proyecto.

Fig. 2.22. Sensor JR3 85M35A3.

3. DESCRIPCIÓN DEL SISTEMA.

3.1. Introducción.

Como se ha descrito en puntos anteriores, la aplicación que se ha desarrollado trata de permitir e interpretar la comunicación entre en sensor de fuerza par JR3 y un robot manipulador ABB.

Mediante la modificación del software proporcionado por JR3 para la adquisición de los datos obtenidos por el sensor se crea una comunicación TCP/IP encargada de enviar los valores recogidos al robot ABB mediante el IRC5. En la Fig 3.1 se muestra el diagrama de bloques del sistema indicando de qué manera se realiza la comunicación.

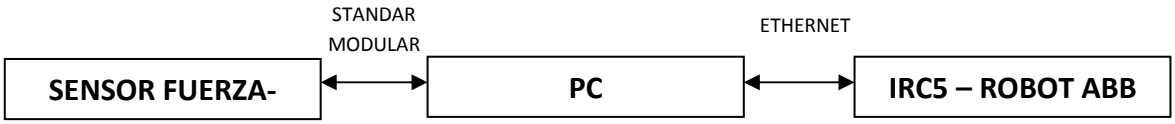


Fig. 3.1. Diagrama de bloques del sistema.

La comunicación entre los distintas partes se pueden dividir en:

- Comunicación mediante cable standar modular para extraer los datos recogidos por el sensor. Esta parte la proporciona JR3 mediante su software.

- Comunicación TCP/IP para enviar los datos al controlador del robot ABB.

Mediante la tarjeta de red de la que dispone el PC y la tarjeta PCI que se instala para recoger los datos del sensor se realizan las comunicaciones necesarias.

3.2. Arquitectura hardware.

En la Fig 3.2 se puede ver representada la arquitectura hardware utilizada en el sistema para realizar las comunicaciones y resolver la aplicación.

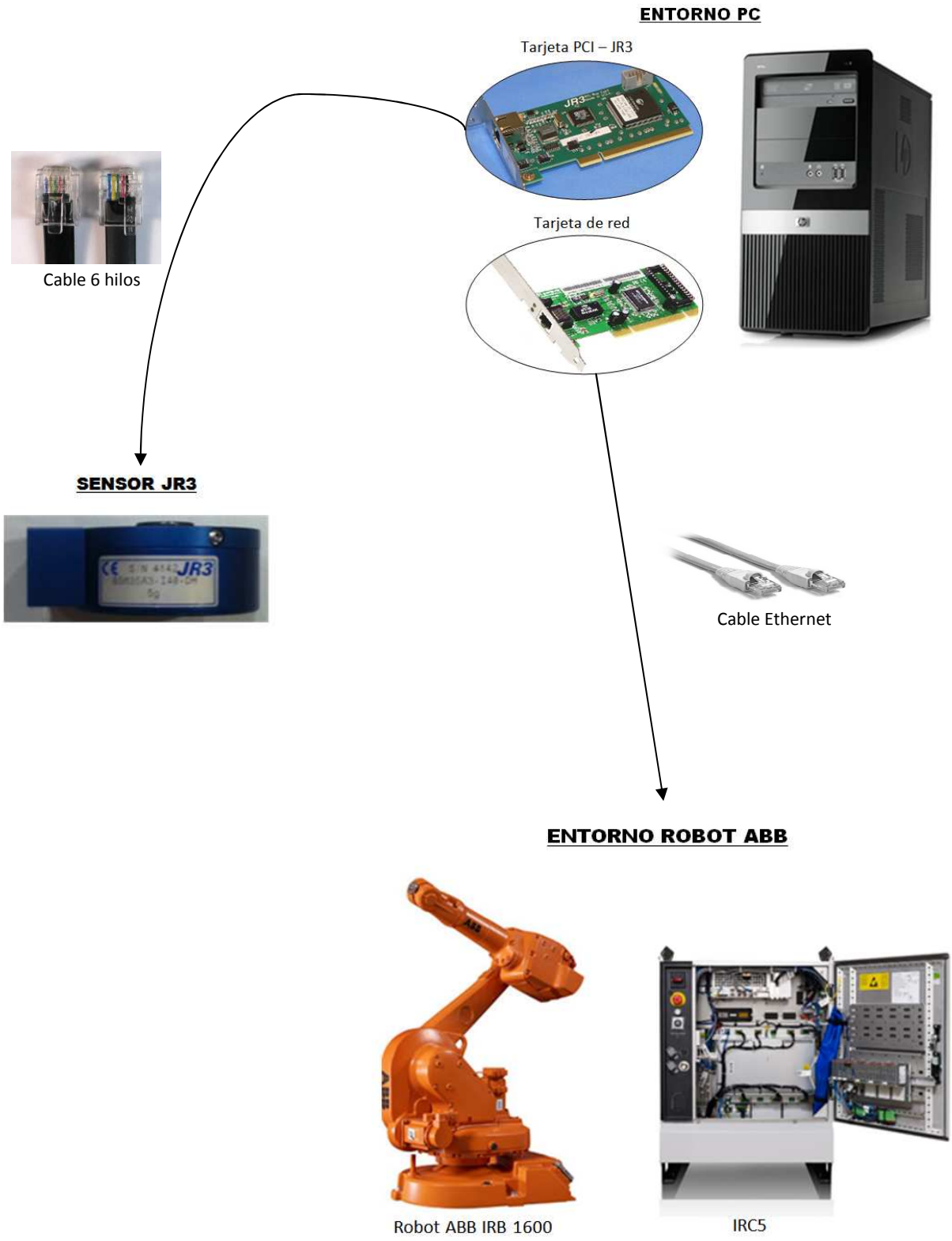


Fig. 3.2. Arquitectura hardware.

3.2.2. Robot ABB IRB 1600.

La célula de trabajo utilizada está formada por un robot ABB IRB 1600 [4]. Se trata de un robot de 6 ejes diseñado específicamente para industrias de fabricación que utilizan la automatización basada en robots. En la Fig.3.3 se muestra el robot ABB IRB 1600.

Fig. 3.3. Robot ABB IRB 1600.

Este robot está diseñado para trabajar de manera óptima en tareas como soldadura por arco, fundición a presión, montaje y embalaje, etc.

Construido en la segunda generación de TrueMove y QuickMove, mejora de manera sustancial la precisión de la trayectoria. También, mediante software se supervisa la carga interna del robot, evitando así situaciones de sobrecarga que permitirá alargar la vida útil del robot.

3.2.2.2 Características generales.

De las características generales del robot ABB IRB 1600 se pueden destacar:

- Su construcción robusta y el uso de pocos componentes hace que sea un robot muy fiable y con pocas necesidades de mantenimiento.
- Debido a mejoras en el software, se ha reducido el tiempo de ciclo con respecto a versiones anteriores.
- Permitiendo una carga útil de entre 6 y 8 kg (hasta 12 kg con muñeca hacia abajo), hacen que el robot IRB 1600 sea el más fuerte de su clase.
- Flexible en su integración y producción permitiéndole así ser instalado en paredes, suelos, posiciones invertidas o inclinadas (véase Fig 3.4).

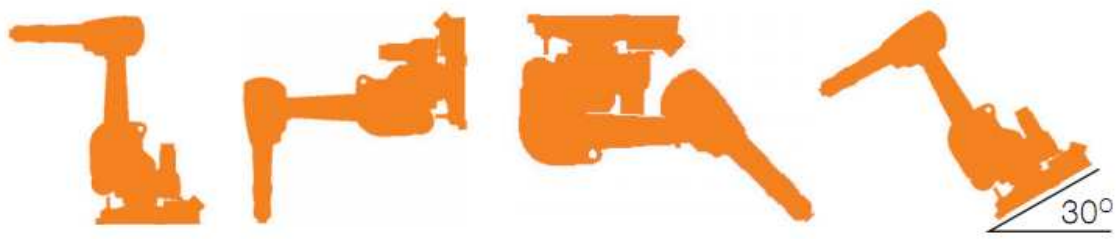


Fig. 3.4. Posiciones de trabajo disponibles para el robot ABB IRB 1600.

3.2.2.3 Ejes del manipulador.

En la siguiente Fig 3.5 se pueden ver cada uno de los 6 ejes del manipulador IRB 1600.

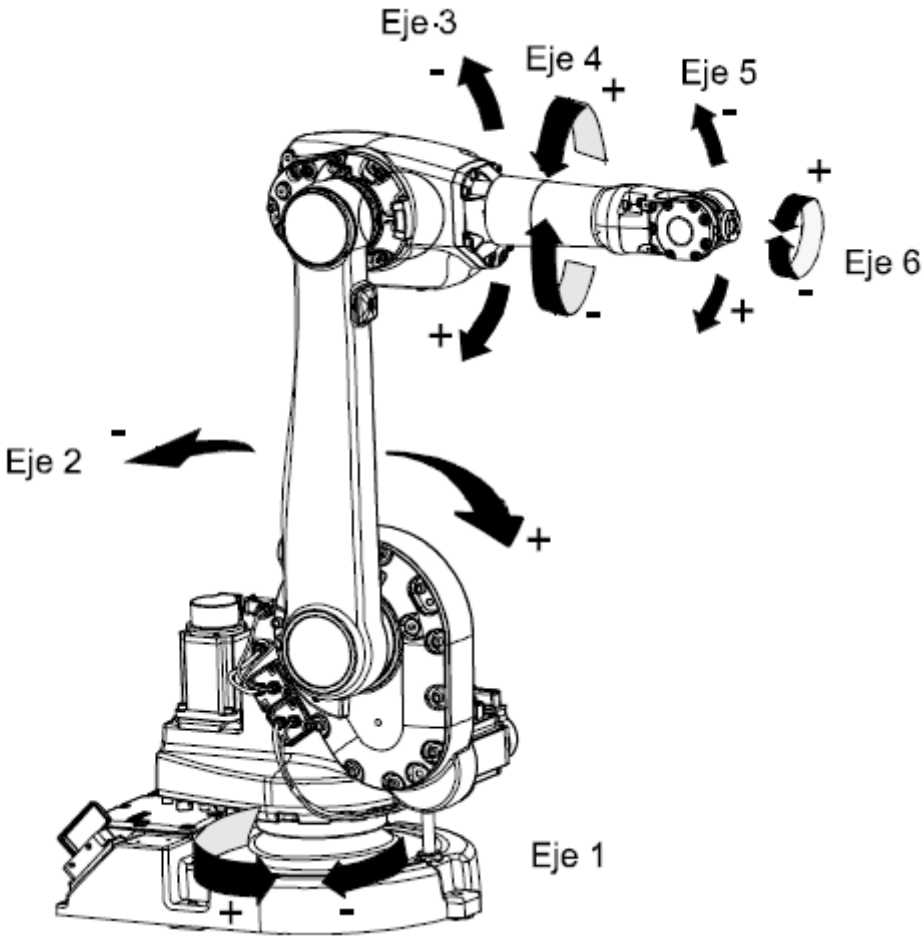


Fig. 3.5. Ejes del manipulador IRB 1600.

3.2.2.4 Dimensiones del IRB 1600.

En la siguiente Fig 3.6 se pueden ver las dimensiones del manipulador IRB 1600.

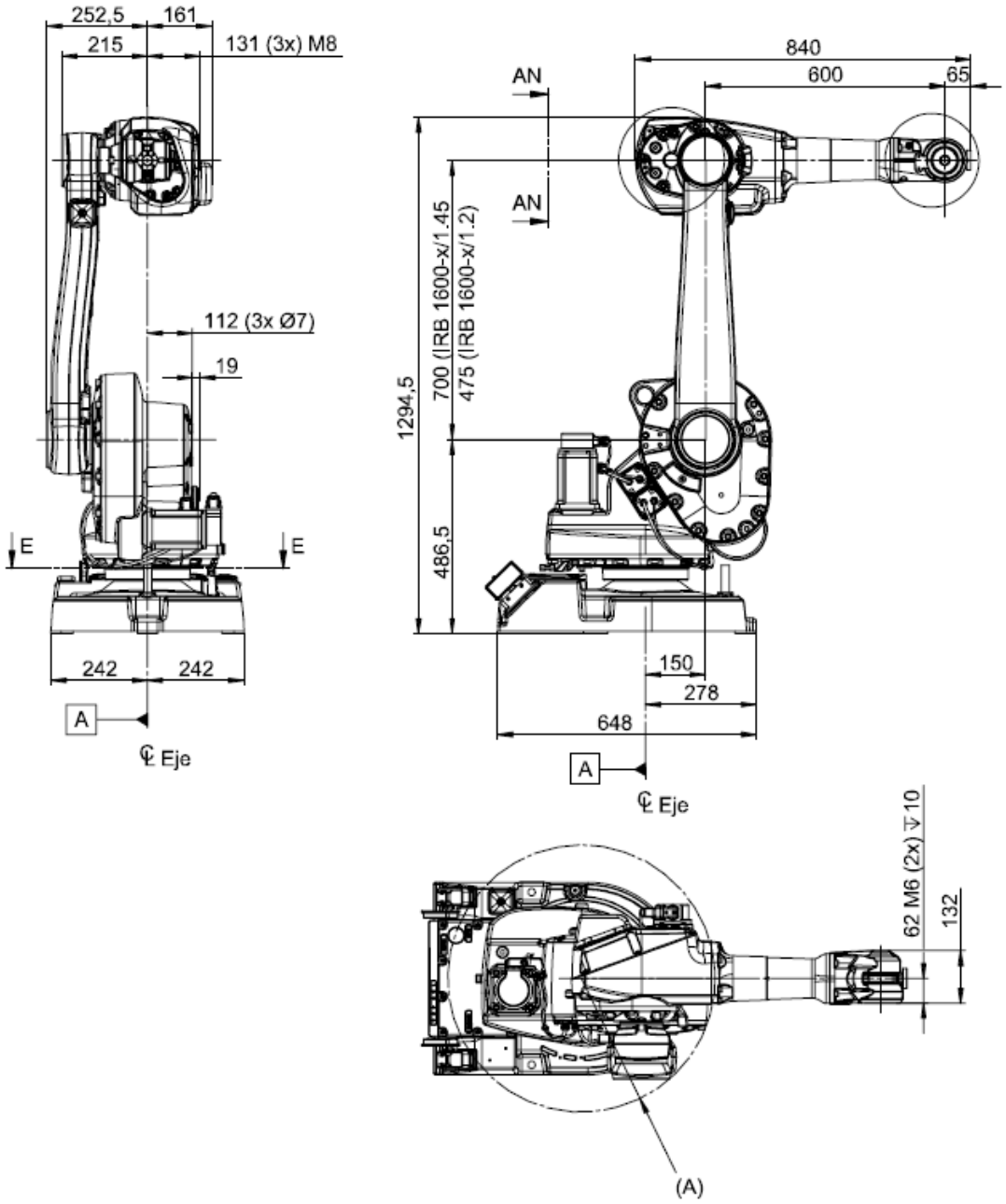


Fig. 3.6. Vista posterior, lateral y superior del manipulador (dimensiones en mm).

3.2.2.5 Rango de movimientos.

En la Tabla 3 1 se muestran los distintos rangos de movimientos que permiten cada uno de los 6 ejes del robot ARR IRB 1600.

Eje	Rango de movimiento
1	+180° a -180°
2	+136° a -63°
3	+55° a -235°
4	+200° a -200°
5	+115° a -115°
6	+400° a -400°

Tabla 3 1. Rangos de movimiento en robot ABB IRB 1600.

3.2.2.6 Velocidad.

La máxima velocidad de giro de cada eje se muestra en la Tabla 3 2.

Eje	Máxima velocidad de giro
1	150°/s
2	150°/s
3	150°/s
4	360°/s
5	360°/s
6	450°/s

Tabla 3 2. Tabla de velocidades máximas del robot.

La supervisión es necesaria para evitar el sobrecalentamiento en aplicaciones con movimientos intensos y frecuentes.

3.2.2.7 Rango de trabajo.

El robot ABB IRB 1600, como cualquier otro tiene unas limitaciones en sus movimientos debido a su morfología. En la Fig 3.7 se pude observar el rango de trabajo de este modelo con todos sus alcances en cada una de sus posiciones extremas.

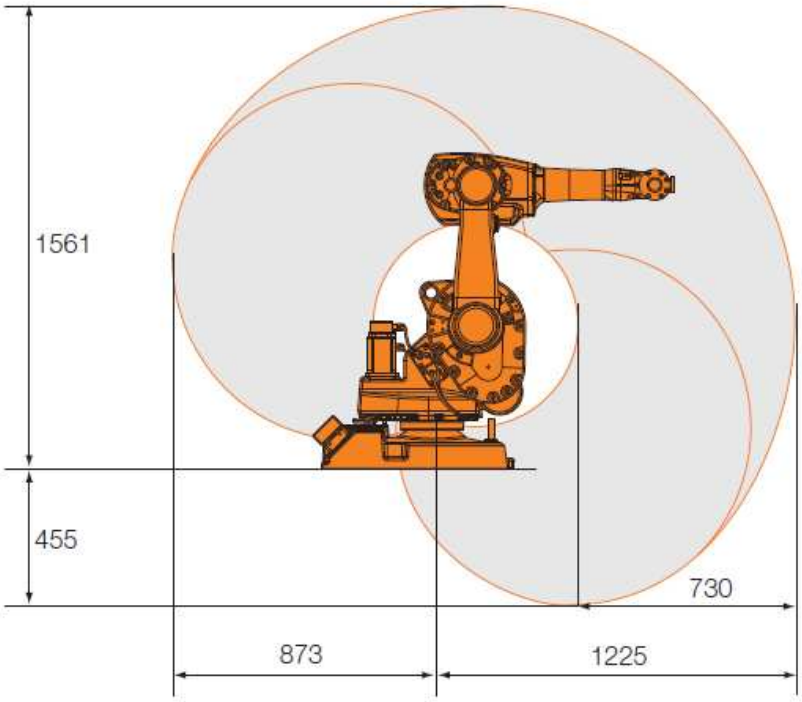


Fig. 3.7. Rango de trabajo – Robot ABB IRB 1600.

3.2.2.8 Diagrama de carga.

Es muy importante definir siempre los datos de carga del robot. Una definición incorrecta de los datos de carga puede dar lugar a la sobrecarga del robot, pudiendo dañar los motores, las cajas reductoras y la estructura mecánica

A continuación, en la Fig 3.8, se muestra el diagrama de carga, que representa el máximo peso permitido para el montaje de una carga en la brida de montaje con distintas posiciones del centro de gravedad. Las distancias están representadas en metros.

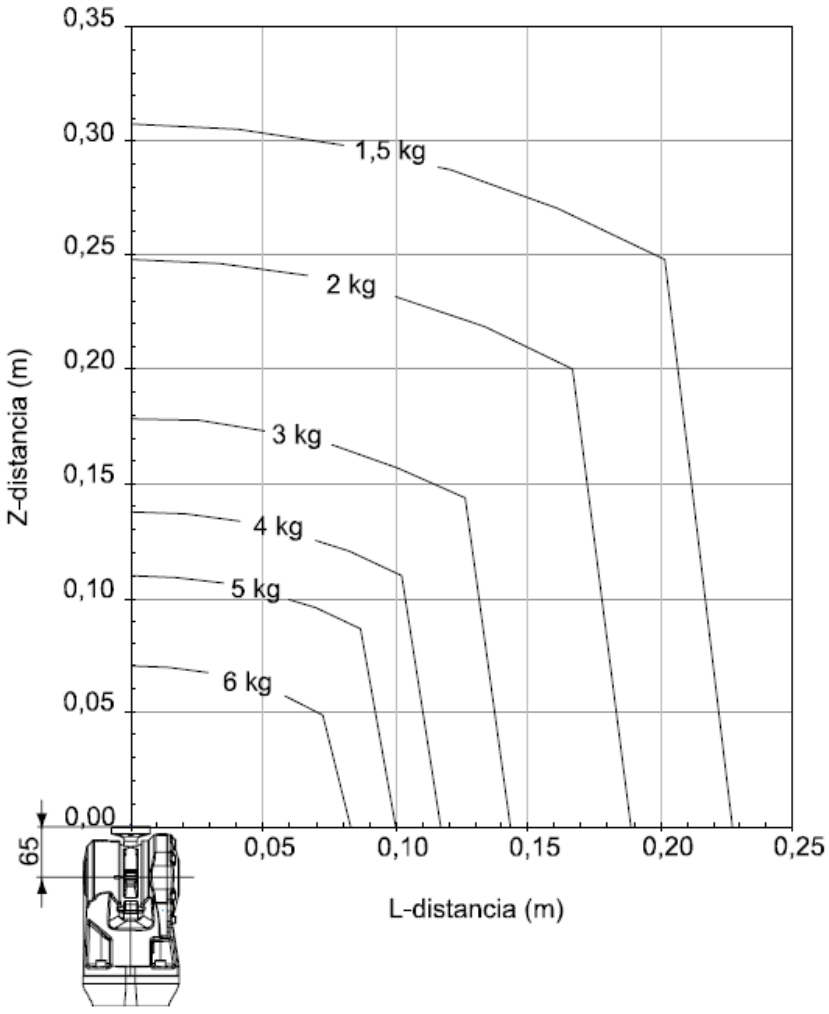


Fig. 3.8. Diagrama de carga.

3.2.2.9 Posiciones extremas del manipulador.

En la Fig 3.9 siguiente se pueden ver las posiciones extremas que permite el robot manipulador ABB IRB 1600.

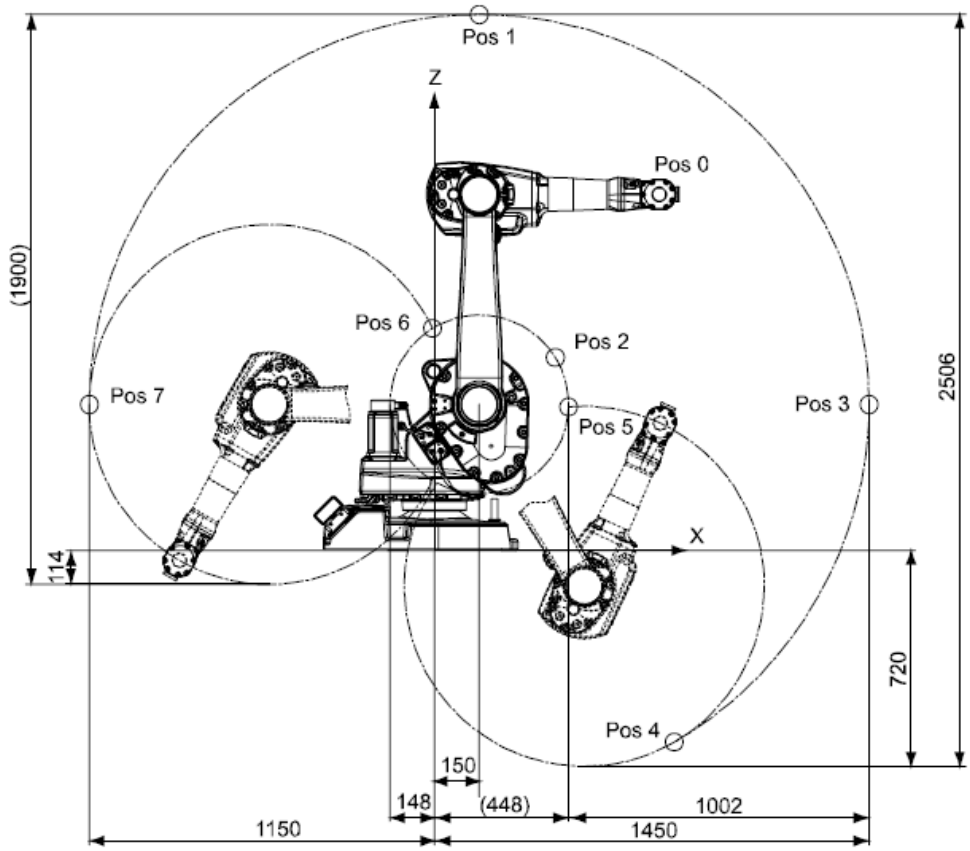


Fig. 3.9. Posiciones extremas del robot.

3.2.2.10 Controlador IRC5 de robot ABB IRB 1600.

El controlador IRC5 es un controlador de robot de ABB de quinta generación, el cual contiene los elementos electrónicos necesarios para controlar el manipulador, los ejes adicionales y los equipos periféricos. Consiste en un único armario de medidas 970 x 725 x 710 mm y 150 kg de peso, que contiene los siguientes módulos:

- **Módulo de accionamiento.**

Contiene el sistema de accionamiento que proporciona la energía necesaria a los motores.

- **Módulo de control.**

Contiene el ordenador, el panel de control, el interruptor de alimentación, las interfaces de comunicación, una tarjeta de entradas y salidas digitales, la conexión para FlexPendant, los puertos de servicio y cierto espacio libre para equipos del usuario. El ordenador cuenta con 4 ranuras PCI para tarjetas de expansión. El controlador también contiene el software de sistema, es decir RobotWareOS, que incluye todas las funciones básicas de manejo y programación (Rapid). Además, el IRC5, cuenta con una unidad de interfaz portátil de diseño ergonómico con pantalla táctil que facilita la comunicación hombre-máquina. Se trata del FlexPendant, el cual se detallará más adelante.

- **Panel de control.**

El panel de control, mostrado en la Fig 3.1, es el elemento donde se localizan tanto los mandos de control del IRC5 accesibles por el usuario, como elementos auxiliares de comunicación temporización, etc.

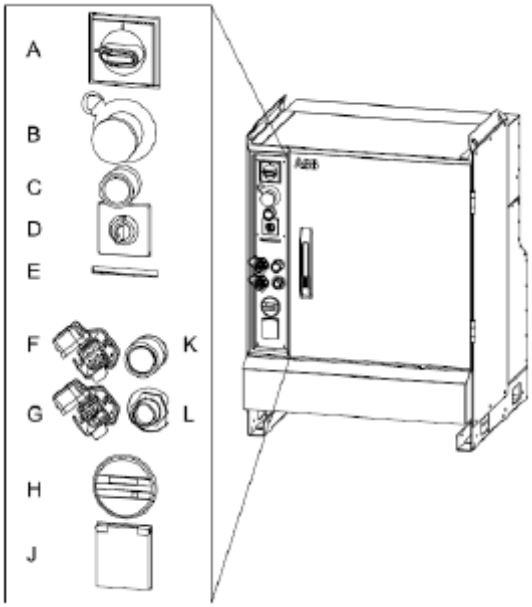


Fig. 3.1. Panel de control del IRC5.

En la siguiente Tabla 33 se muestra una lista de los elementos señalados en la Fig 3.1 y posteriormente se procederá a su explicación detallada:

A	Interruptor principal
B	Paro de emergencia
C	Motores ON
D	Selector de modo
E	LEDs de la cadena de seguridad (opción)
F	Puerto USB (opción)
G	Puerto de servicio para PC (opción)
H	Contador de tiempo de funcionamiento (opción)
J	Toma de servicio 115/230 V, 200 W (opción)
K	Botón de conexión en caliente (opción)
L	Conector para FlexPendant

Tabla 3 3. Definición panel de control IRB5.

- **Motores ON.**

Es una luz que nos indica el estado de los motores. Si está encendida permanentemente, quiere decir que está listo para la ejecución de programas. Si parpadea quiere decir que se ha activado uno de los paros de protección, el robot no está calibrado o los cuentarrevoluciones no están actualizados.

- **Modo de funcionamiento.**

Mediante un interruptor con llave, es posible poner el robot en tres modos de funcionamiento distintos:

- Modo manual a velocidad reducida: Programación y configuración. Velocidad máxima 250mm/s. Este modo de funcionamiento se emplea cuando se está programando el robot, creando puntos y trayectorias, para lo cual puede ser necesaria la presencia del operario cerca del robot.

- Modo manual a máxima velocidad: comprobación a velocidad máxima del programa. Una vez verificado el correcto funcionamiento del programa, se debe realizar una última prueba a la velocidad real del programa.

- Modo automático: Producción en funcionamiento. Una vez superada la prueba del modo manual a máxima velocidad, el robot está preparado para funcionar autónomamente en el modo automático.

- **Conexión de PC de servicio.**

Mediante este conector Ethernet conectaremos el IRC5 a un PC mediante la herramienta Robotstudio como veremos más adelante.

- **Conexión USB.**

Mediante este conector podremos cargar programas en el controlador usando una memoria extraíble.

3.2.2.11 FlexPendant.

El FlexPendant es el elemento encargado de comunicar al hombre con la máquina y viceversa. Consiste en un mando, como el mostrado en la siguiente imagen, con una pantalla táctil y distintos botones con los que poder programar, configurar, e incluso monitorizar el estado del robot como se puede ver en la Fig 3.2.

Fig. 3.2. Flexpendant.

En la siguiente Tabla 3 4 se muestra una lista de los elementos señalados en la Fig 3.2 y posteriormente se procederá a su explicación detallada:

Elemento	Descripción
A	Pantalla táctil
B	Teclas definidas por el usuario
C	Pulsador de paro de emergencia
D	Joystick
E	Teclas de ejecución de programa
F	Dispositivo de habilitación

Tabla 3 4. Definición botones Flexpendant.

A. Pantalla táctil.

Pantalla táctil en color de 7,5 pulgadas que muestra textos e información gráfica. Es posible tener abiertas varias ventanas a la vez. En muchas vistas es posible ampliar y reducir la imagen. Muchas propiedades de la pantalla pueden ser configuradas por el usuario para conseguir un aspecto y una funcionalidad personalizados. Es posible invertir la pantalla y las direcciones del joystick para hacer que sea adecuado para usuarios zurdos.

B. Teclas definidas por el usuario.

Cuatro teclas definidas por el usuario que pueden configurarse para activar o restablecer una salida (por ejemplo para abrir o cerrar una pinza).

C. Pulsador de paro de emergencia:

Cuando se presiona este botón, el robot se detiene inmediatamente.

D. Joystick.

El joystick tridimensional se utiliza para mover el robot manualmente, por ejemplo durante los trabajos de programación. El usuario puede determinar la velocidad de este movimiento. Una inclinación pronunciada del joystick hace que el robot se mueva rápidamente. Una inclinación más leve hará que se mueva más lentamente.

E. Teclas de ejecución de programas.

Teclas para iniciar y detener programas y ejecutarlos paso a paso hacia delante o hacia atrás.

F. Dispositivo de habilitación.

Durante el modo manual, un pulsador que, si está presionado hasta la mitad de su recorrido, pone el sistema en el modo MOTORES ON. Cuando se libera el dispositivo de habilitación o si es pulsado totalmente, el robot pasará al estado MOTORES OFF.

3.2.2.11.1 Detalle del menú del FlexPendant.

- **Barra de estado.**

En la Fig 3.3 se muestra la barra de estado de FlexPendant.

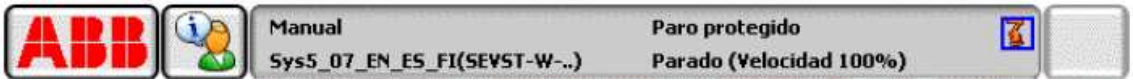


Fig. 3.3. Barra de estado – FlexPendant.

En ella se muestra el modo de funcionamiento (Auto, Manual, Manual a velocidad máxima), el nombre del controlador, el estado de los motores y el estado de ejecución en el que se encuentra el robot, parado (Stopped) o en movimiento (Running), junto con la velocidad de ejecución.

- **Menú de configuración rápida.**

En el menú de configuración rápida, mostrado en la Fig 3.4, se puede modificar la velocidad, el modo de ejecución (Un solo ciclo o Continuo), incrementos en los movimientos, etc.

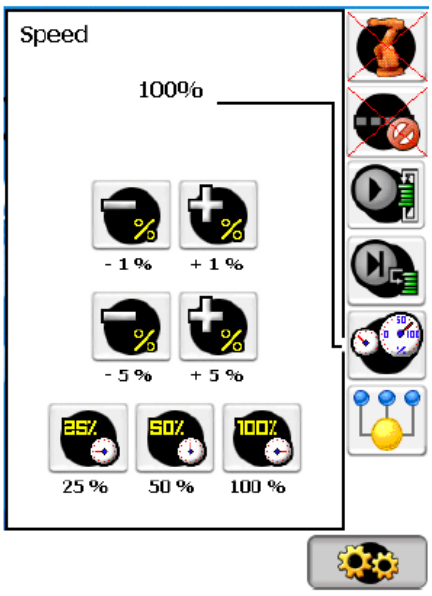


Fig. 3.4. Menú de configuración rápida.

- **Menú ABB.**

En la Fig 3.5 se muestra el Menú ABB en el cual se pueden ver y simular las Entradas/Salidas, mover los ejes del robot, editar los programas RAPID y los datos de programa, realizar backups y restaurar desde éstos, calibrar el robot actualizando el cuentarrevoluciones, tener acceso al panel de control y al explorador de archivos de FlexPendant.

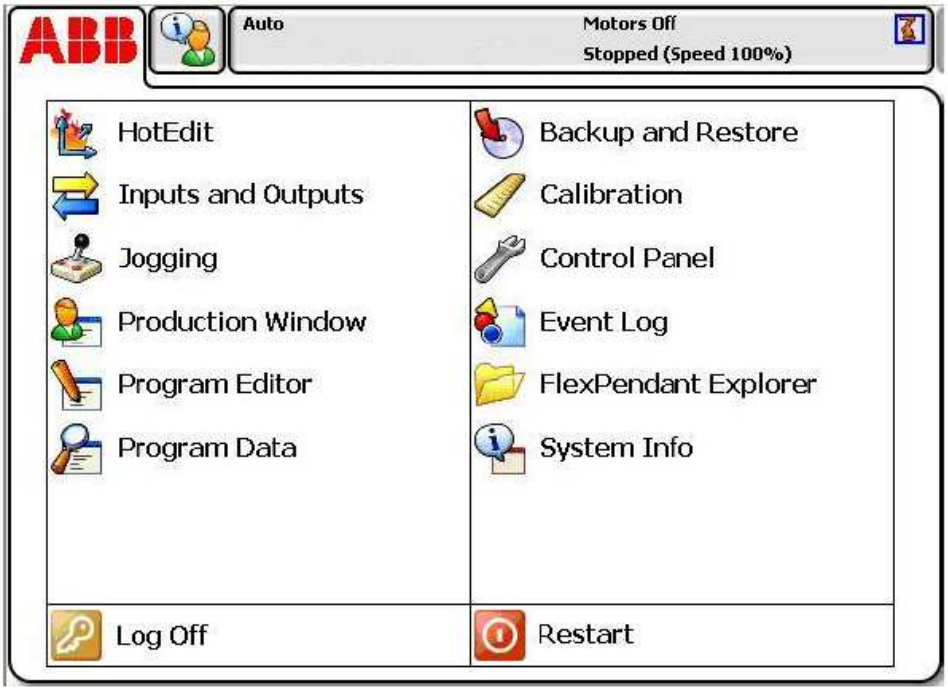


Fig. 3.5. Menú ABB.

Otra función importante del Menú ABB es el Reinicio. Hay muchos tipos de reinicio:

- Arranque W (reinicio en caliente) – utilizado para reiniciar y seguir usando el sistema actual. Todos los parámetros del sistema y programas se guardan en un archivo de imagen. Durante el proceso de reinicio, se reanuda el estado del sistema. Las tareas estáticas y semiestáticas se inician. Es posible iniciar los programas desde el punto en que fueron detenidos.

- Arranque C (reinicio en frío) – Si se quiere cambiar a otro sistema instalado o instalar un nuevo sistema y, al mismo tiempo, eliminar de la memoria el sistema actual. Todo el contenido, las copias de seguridad y los programas del directorio del sistema se eliminarán. Esto significa que será imposible reanudar el estado de este sistema de ninguna forma.

- P-Start - Elimina todos los programas de RAPID cargados por el usuario.

- I-Start - Reinicio de la instalación.

- B-Start - Reinicio desde datos de sistema almacenados previamente.

3.2.3. Sensor JR3 85M35A3.

Se trata de un sensor de fuerza par fabricado principalmente de aluminio aunque también se puede encontrar en acero inoxidable o titanio. Contiene sistemas analógicos y digitales que forman su electrónica interna así como las galgas encargadas de medir la deformación en cada uno de los ejes. Las señales leídas son amplificadas para convertirse en representaciones analógicas. La mayoría de los modelos tratan esta información analógica convirtiéndola en digital para su mejor interpretación. En la Fig 3.6 se muestra el sensor JR3 utilizado.

Fig. 3.6. Sensor JR3 85M35A3.

Los distintos modelos proporcionados por el fabricante permiten ejercer sobre ellos diferentes esfuerzos de carga. Dependiendo de los esfuerzos que el sensor tenga que soportar, JR3 tiene a disposición de sus clientes numerosos modelos de mayor y menor tamaño que permiten ajustarse a cada una de las necesidades.

En la Fig 3.7 se puede ver la orientación de los ejes en este tipo de sensores.

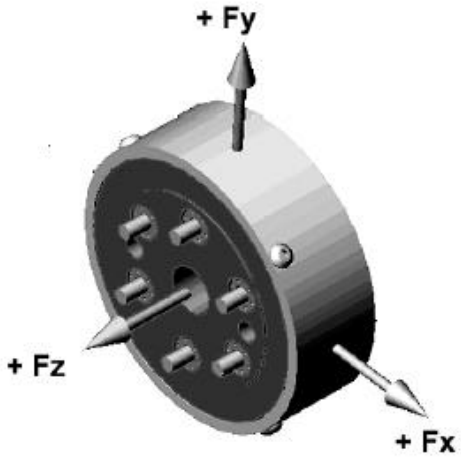


Fig. 3.7. Dirección de los ejes en sensor JR3 85M35A3.

3.2.3.2 Dimensiones.

En la Fig 3.8 se pueden observar las medidas del sensor JR3 modelo 85M35A3. Los valores de medida los proporciona el fabricante y en este caso se encuentran en pulgadas.

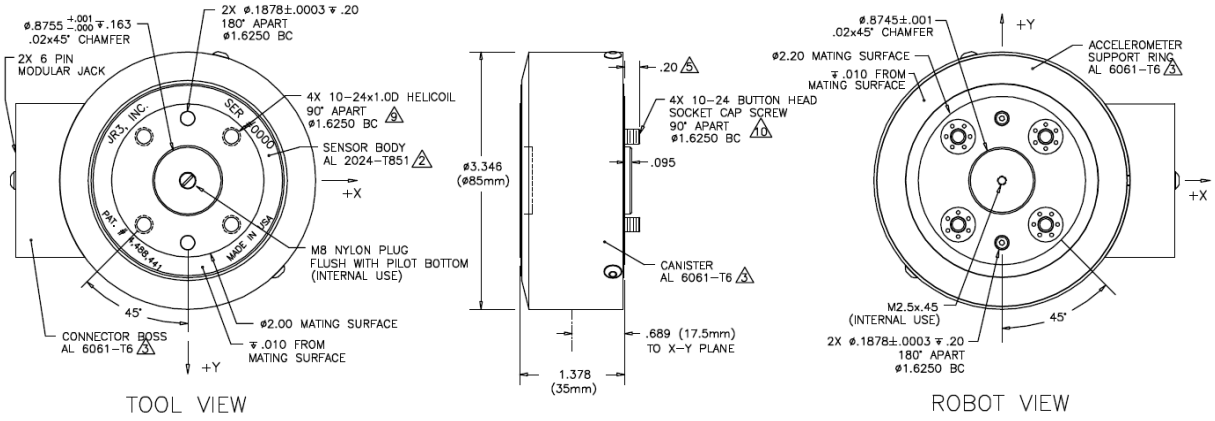


Fig. 3.8. Dimensiones sensor JR3 85M35A3.

3.2.3.3 Interfaz electrónico

El sensor de JR3 que se utiliza en este proyecto proporciona una salida digital en un formato síncrono. Todas las señales analógicas así como el A/D esta incluidos en el interior del cuerpo del sensor, protegido de las interferencias electromagnéticas gracias al componente metálico de su carcasa. Los datos que se generan por la acción de las cargas se transmiten a 8 KHz, pudiendo otros sensores de la gama transmitir a otras frecuencias. En la Fig 3.9 se observa el cable utilizado para comunicación del sensor con el PC.

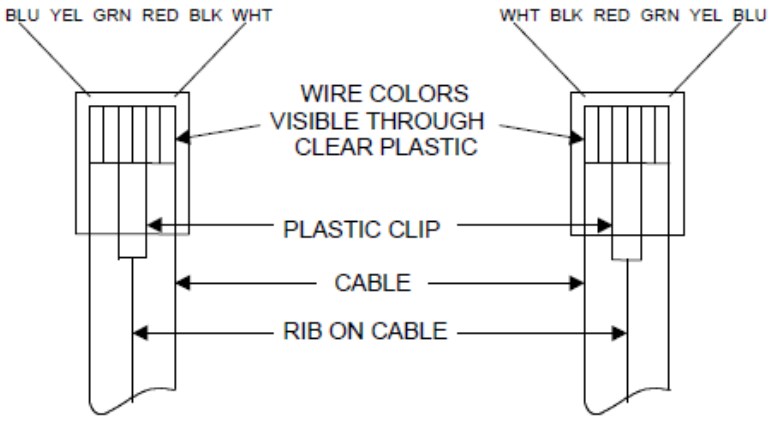


Fig. 3.9. Cable del sensor.

No obstante, además de los datos también se envía información de la corriente de alimentación del sensor, así como información de las características del sensor y de la calibración, lo que permite no tener que estar calibrando o ajustando el circuito receptor. En el caso del sensor que se está utilizando, el conector es un RJ-11 de 6 pins como el de la Fig 3.9.

La asignación de pins es tal y como se indica en la Tabla 35.

PIN	SIGNAL
1	DCLK +
2	DCLK -
3	+ ~ 8 V
4	PWR COM
5	DATA +
6	DATA -

Tabla 3 5. Asignación de pins.

Como es habitual en los sistemas electrónicos la tierra del sensor y la del sistema de adquisición de datos debe ser la misma, para evitar que el no compartir tierra pueda ocasionar problemas a la hora de interpretar los datos.

Existe una amplia gama de sistemas de recepción y procesamiento de datos que reciben los datos en serie desde el sensor y proporcionan los valores de la carga en el formato que se desee. Existen diferentes sistemas que permiten una conexión directa con el ordenador ya sea por los puertos serie o paralelo.

3.2.3.4 Matriz de calibración.

Todos los sensores multiejes tienen un cierto grado de acoplamiento, lo que quiere decir que un momento o una carga en un determinado eje produce una cierta carga en los otros ejes. Cada sensor de JR3 está calibrado individualmente con cargas aplicadas en cada eje.

Los datos de calibración se utilizan para generar una matriz de calibración y desacoplamiento, que es utilizada para convertir los voltajes de salida en una fuerza o en un momento en las unidades deseadas.

En el caso del sensor que se está utilizando, al ser de salida digital, almacena los datos de salida en una memoria no volátil, y cuando el sensor es conectado al receptor, los datos son transferidos automáticamente descargándose en los primeros segundos de operación. Después de eso el receptor aplica la matriz de calibración a los datos que se van recibiendo sin intervención del usuario.

La matriz de calibración se aplica tal y como se indica en la Fig 3.10.

$$\begin{matrix} Fx \\ Fy \\ Fz \\ Mx \\ My \\ Mz \end{matrix} \begin{bmatrix} A1,1 & A1,2 & A1,3 & A1,4 & A1,5 & A1,6 \\ A2,1 & A2,2 & A2,3 & A2,4 & A2,5 & A2,6 \\ A3,1 & A3,2 & A3,3 & A3,4 & A3,5 & A3,6 \\ A4,1 & A4,2 & A4,3 & A4,4 & A4,5 & A4,6 \\ A5,1 & A5,2 & A5,3 & A5,4 & A5,5 & A5,6 \\ A6,1 & A6,2 & A6,3 & A6,4 & A6,5 & A6,6 \end{bmatrix} X \begin{bmatrix} C1 \\ C2 \\ C3 \\ C4 \\ C5 \\ C6 \end{bmatrix} = \begin{bmatrix} Fx \\ Fy \\ Fz \\ Mx \\ My \\ Mz \end{bmatrix}$$

Fig. 3.10. Matriz de calibración.

En el caso de resolver la ecuación manualmente sería más complicado dado que habría que resolver las ecuaciones una a una.

3.3. Arquitectura Software.

3.3.1. Introducción.

Para llevar a cabo el desarrollo del proyecto se han utilizado el siguiente software:

- Para la modificación y desarrollo del software encargado de la adquisición, trato y envío de los datos al controlador del robot ABB se ha utilizado Visual Studio 2008 (C++).
- Para el diseño de las piezas de sujeción del sensor al robot se utiliza el programa de diseño Solid Edge.

- Para el desarrollo del software encargado la recepción y trato de los datos asi como la realización de los movimientos en el controlador del robot ABB se ha utilizado RobotStudio.

- Por último, nombrar el software base utilizado que proporciona JR3 para la lectura de los datos del sensor fuerza-par.

3.3.2. Visual Studio 2008

Microsoft Visual Studio 2008 es un entorno de desarrollo integrado (IDE) para sistemas operativos Windows que permite crear aplicaciones, sitios y servicios web.

Este programa se puede encontrar en diferentes versiones diferenciándose en el lenguaje que utiliza. Principalmente se pueden utilizar los siguientes:

- Visual Basic Express Edition.
- Visual C# Express Edition.
- Visual C++ Express Edition.

Para la realización de este proyecto de dispuso del Microsoft Visual C++ Express Edition 2008.

La gama de productos de Visual Studio comparte un único entorno de desarrollo integrado (IDE) que se compone de varios elementos: la barra de herramientas Menú, barra de herramientas Estándar, varias ventanas de herramientas que se acoplan u ocultan automáticamente a la izquierda, parte inferior y a la derecha, así como en el espacio del editor. Las ventanas de herramientas, menús y barras de herramientas disponibles dependen del tipo de proyecto o archivo en el que esté trabajando.

3.3.2.1 Interfaz.

Dependiendo de la configuración aplicada y de las subsiguientes personalizaciones que haya realizado, variará la colocación de las ventanas de herramientas y de otros elementos en el IDE [9].

En la Fig 3.11 se muestra el menú de inicio y en la Fig 3.12 un ejemplo de la pantalla principal.

Fig. 3.11. Menú de inicio.

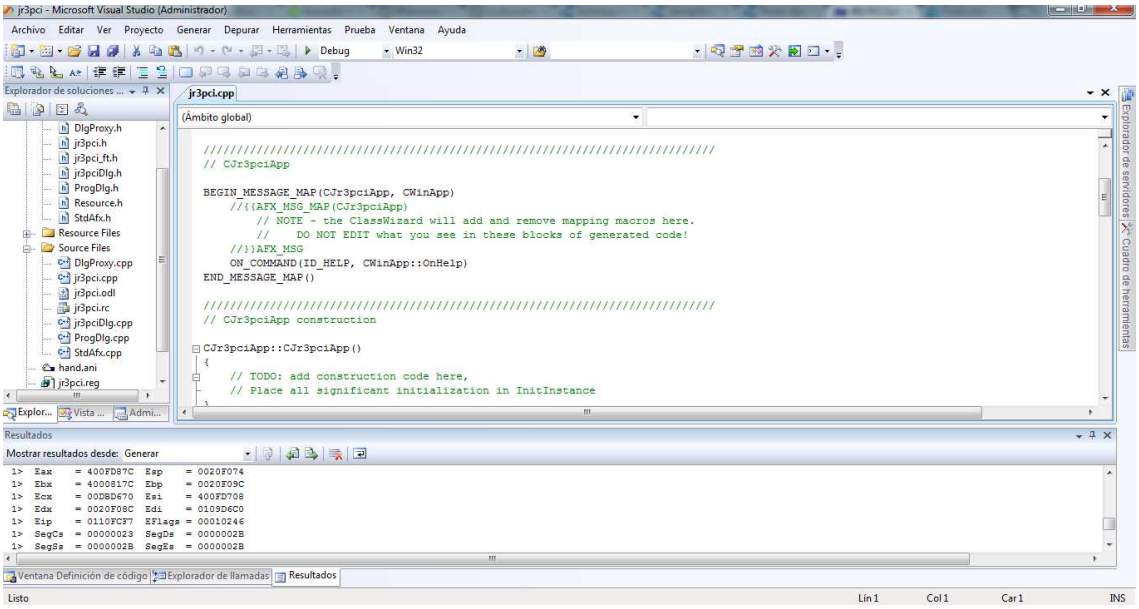


Fig. 3.12. IDE con la configuración de desarrollo general aplicada.

3.3.2.2 Sistema de proyectos.

Las soluciones y los proyectos contienen elementos en forma de referencias, conexiones de datos, carpetas y archivos necesarios para crear la aplicación. Un contenedor de tipo solución puede contener varios proyectos y un contenedor de tipo proyecto normalmente contiene varios elementos.

El Explorador de soluciones; definido en la Fig 3.13 muestra soluciones, proyectos y los elementos incluidos en dichos proyectos. En el Explorador de soluciones, se pueden abrir archivos para editar, agregar nuevos archivos a un proyecto y ver las propiedades de las soluciones, proyectos y elementos.

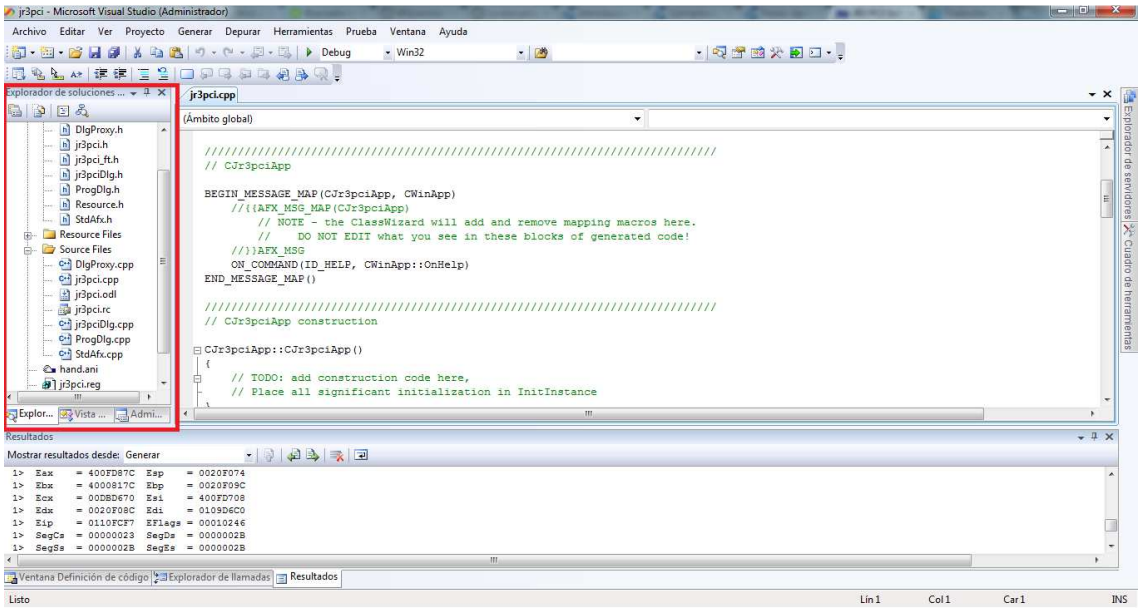


Fig. 3.13. Explorador de soluciones

3.3.2.3 Editores y diseñadores.

El editor y los diseñadores que utilicen dependerán del tipo de archivo o documento que esté creando. El Editor de texto es el procesador de textos básico del IDE, mientras que el Editor de código es el editor de código fuente básico.

Otros editores, como el Editor CSS, el Diseñador HTML y el Diseñador de páginas Web, comparten muchas de las funciones del Editor de código, junto con mejoras específicas en el tipo de código o de marcado admitido.

Los editores y diseñadores normalmente tienen dos vistas: una vista de código subyacente o vista de código fuente (Fig 3.14) y una vista de diseño gráfica (Fig 3.15). La vista de diseño permite especificar la ubicación de los controles y otros elementos en la interfaz de usuario o la página web. Para este proyecto esta opción no se utiliza por no necesitar de su uso.

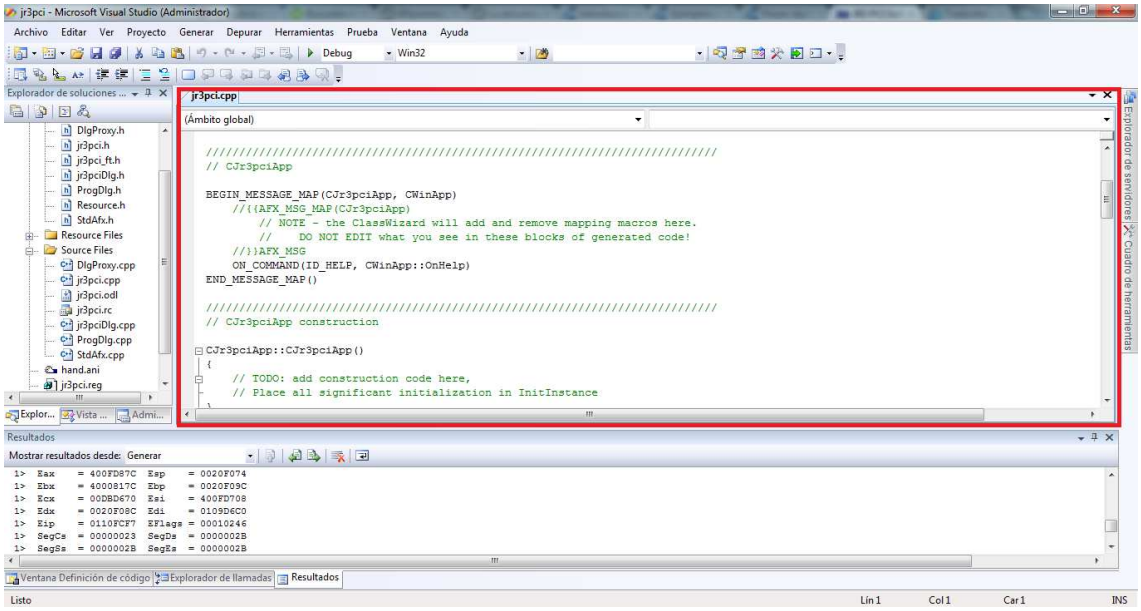


Fig. 3.14. Vista de Código Fuente.

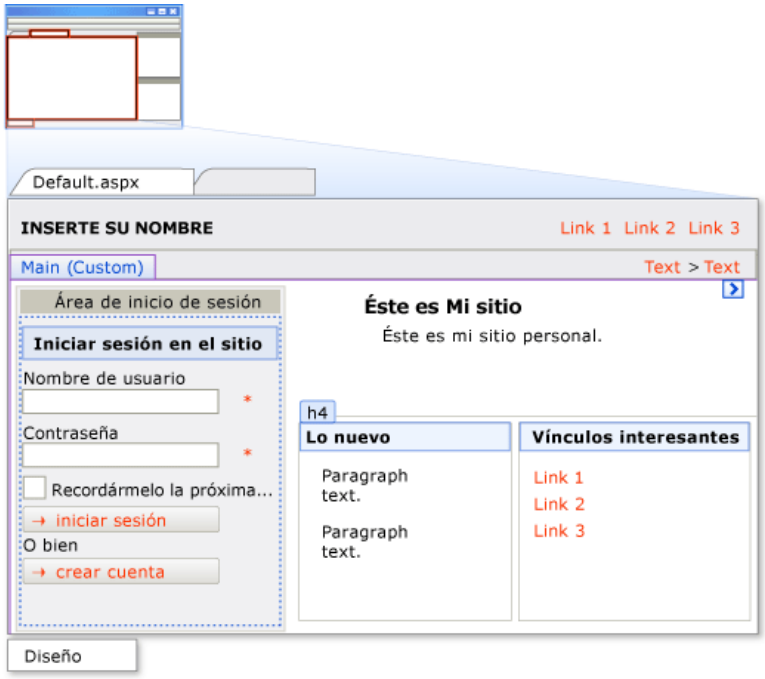


Fig. 3.15. Diseñador de páginas Web, vista Diseño

También se puede utilizar una vista híbrida que permite ver la vista del gráfico y del código de un archivo simultáneamente. Esta vista se llama la Vista dividida y se puede ver en la Fig 3.16.

Fig. 3.16. Diseñador de páginas web, Vista dividida.

3.3.2.4 Herramientas de compilación y depuración.

Visual Studio proporciona un conjunto de herramientas de compilación y depuración. Con las configuraciones de compilación puede seleccionar los componentes que se van a generar, excluir los que no se van a generar y determinar cómo se van a generar los proyectos seleccionados y en qué plataforma.

Cuando genera, está comenzando el proceso de depuración. Durante la depuración se pueden detectar errores de compilación, como se puede ver en la Fig 3.17.

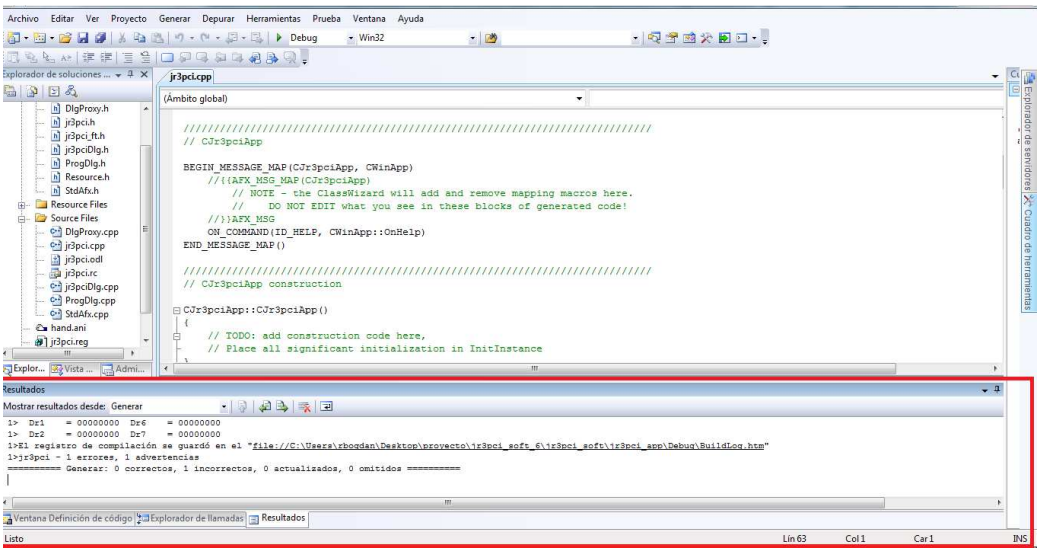


Fig. 3.17. Ventana Resultados con información de compilación.

Después de generar la aplicación, puede utilizar el depurador para detectar y corregir problemas como errores lógicos y semánticos que se descubren en tiempo de ejecución.

Visual Studio se puede compilar en el modo de interrupción, en el cual se puede examinar las variables locales y otros datos pertinentes utilizando puntos de ejecución paso a paso mostrados en la Fig 3.18.

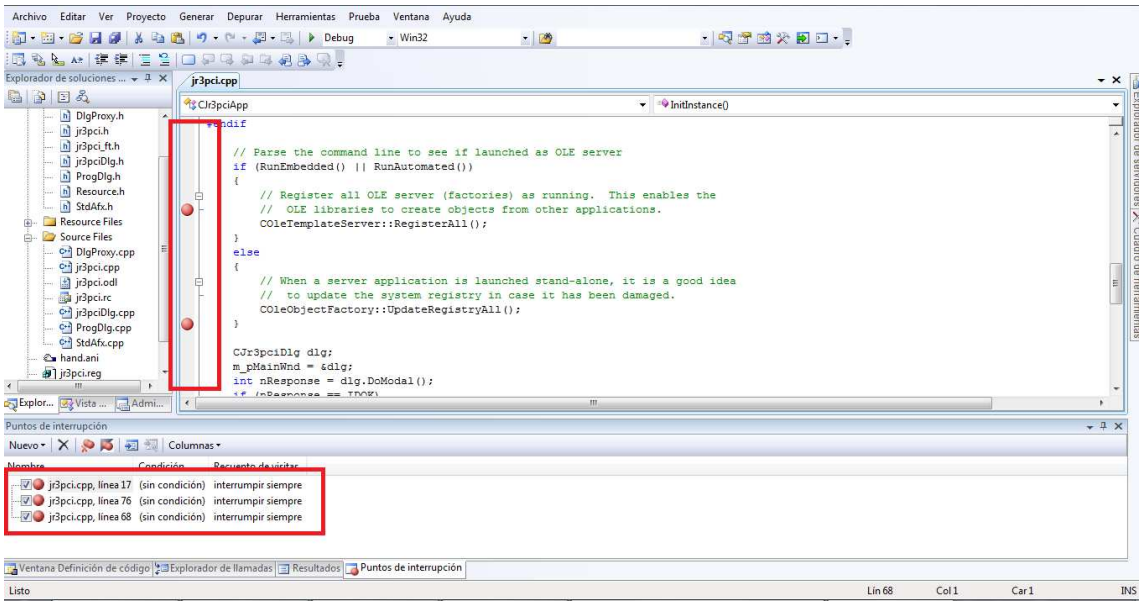


Fig. 3.18. Formulario de Visual Basic en el modo de interrupción.

3.3.2.5 Herramientas de implementación.

Visual Studio proporciona dos estrategias de implementación diferentes: ClickOnce y Windows Installer. Con la implementación de ClickOnce, publica la aplicación en una ubicación centralizada y el usuario instala o ejecuta la aplicación desde esa ubicación. La implementación de Windows Installer (Fig 3.19) permite empaquetar la aplicación en un archivo setup.exe y distribuir ese archivo; pudiendo ejecutar el archivo setup.exe para instalar la aplicación.

En este proyecto se utilizara la última estrategia de implementación, por la facilidad que supone distribuir el programa.

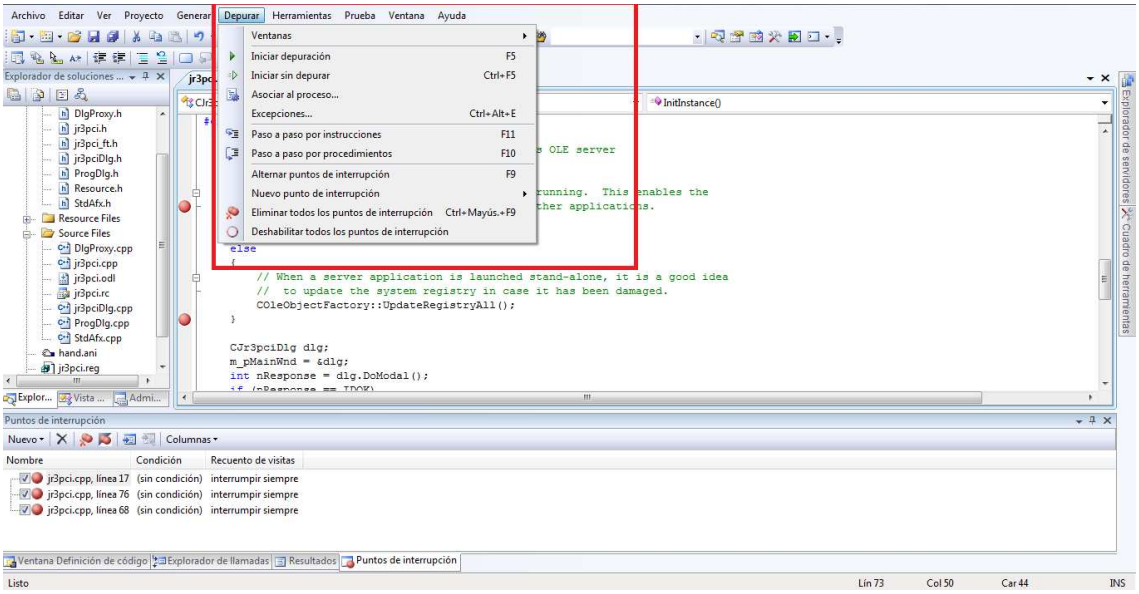


Fig. 3.19. Menú para generar archivo Windows installer.

3.3.3. Solid Edge.

3.3.3.1 Introducción.

Se trata de un programa parametrizado de diseños asistido por computador de piezas tridimensionales. Permite diseñar piezas de distintos materiales, doblado de chapas, ensamblaje de conjuntos, representación de soldaduras, generación de planos en 2D.

El usuario podrá encontrar un índice en el cual se diferencia claramente entre los diferentes entornos disponibles de Solid Edge: Pieza, Chapa, Conjunto, Plano y Soldadura.
A continuación se explicarán las principales funciones y comandos de Solid Edge que se pueden usar en el entorno en el que se desarrolla el proyecto (pieza).

3.3.3.2 Plantillas de Solid Edge.

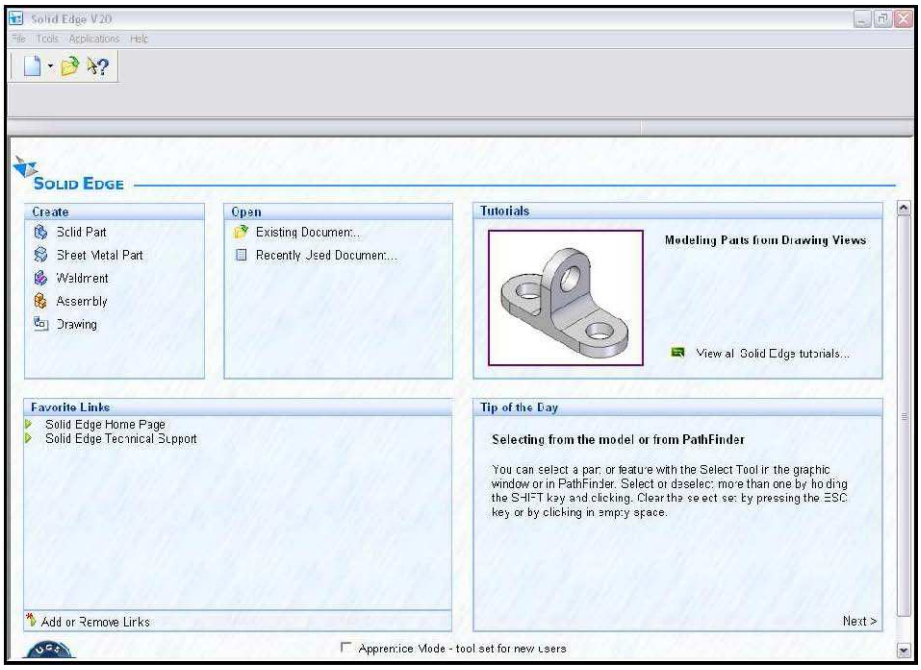


Fig. 3.20. Página de inicio Solid Edge.

Solid Edge dispone de una serie de plantillas predeterminadas, para trabajar con herramientas específicas en cada una de ellas. Las plantillas no son sino ficheros vacíos a los que el usuario añadirá geometrías para guardarlas luego con otros nombres.

Estas aparecen en la pestaña general del cuadro de diálogo de NUEVO, en la ventana de inicio STARTUP o en el desplegable del icono NUEVO de la barra de herramientas principal; presentes en la figura 3.20.

En la figura 3.21, mostrada a continuación, se visualizan las distintas plantillas que hacen referencia a los entornos de trabajo.

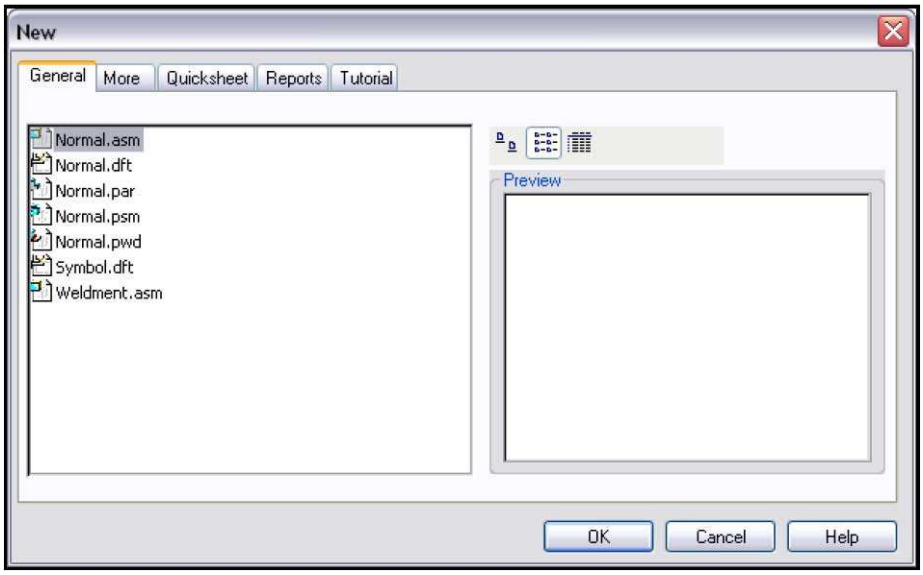


Fig. 3.21. Distintas plantillas de trabajo.

Las diferentes plantillas son las siguientes:

NORMAL.PAR	Documentos de Pieza
NORMAL.PSM	Documentos de Chapa
NORMAL.DFT	Documentos de Plano
NORMAL.PWD	Documentos de Soldadura
NORMAL.ASM	Documentos de Conjuntos

Además de los formatos propios de SE, se puede trabajar con archivos de otros programas (por ejemplo *.dwg y *.dxf de AutoCad).

3.3.4. Descripción de la interfaz.

3.3.4.1 Entorno grafico.

El interfaz contiene una barra de menús, una barra de herramientas y una barra específica de cada comando llamada cinta. Contiene además otras herramientas que ayudan al usuario a conocer los comandos, manipular las ventanas, especificar los colores y seleccionar elementos. En la figura 3.22 son mostrados los diferentes elementos, y a continuación serán definidos.

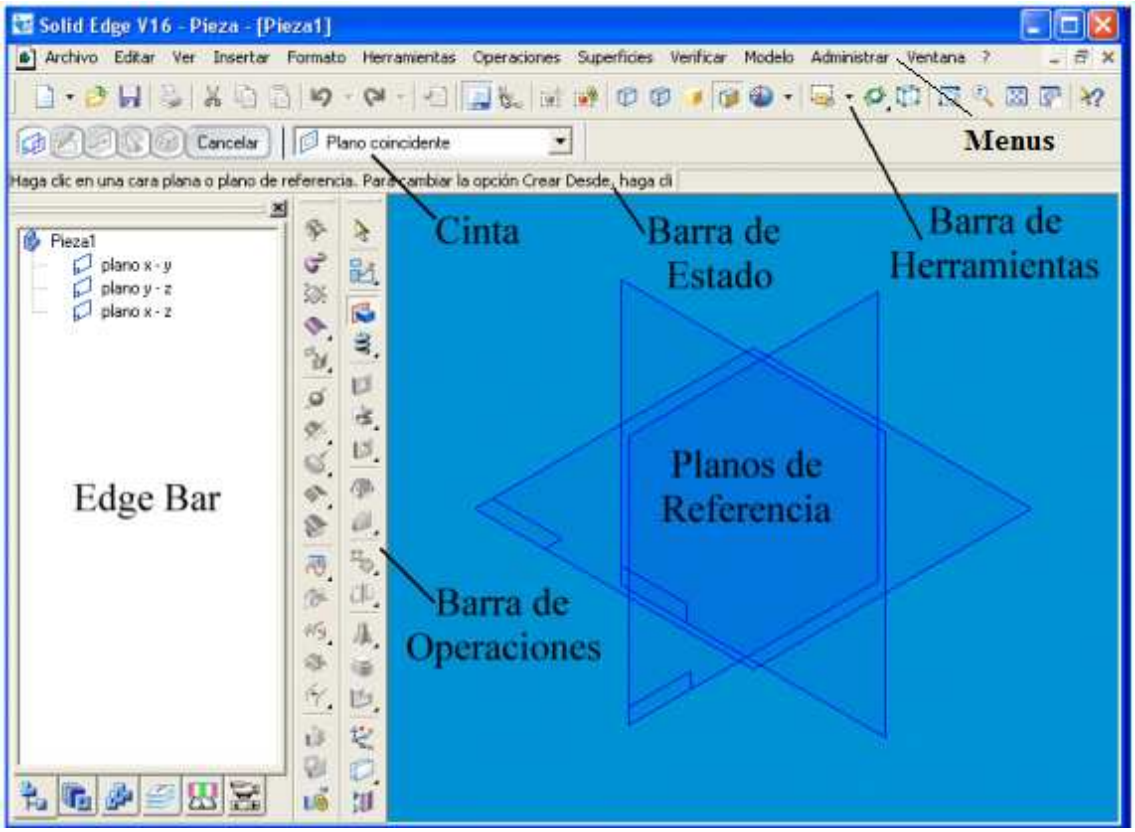


Fig. 3.22. Entorno grafico Solid Edge.

3.3.4.2 Planos de referencia

Cuando se abre una plantilla nueva de PIEZA, que es el entorno que se trata en este proyecto, en el área de trabajo se visualizan 3 planos de referencia posicionados ortogonalmente entre ellos como en la figura 3.23.

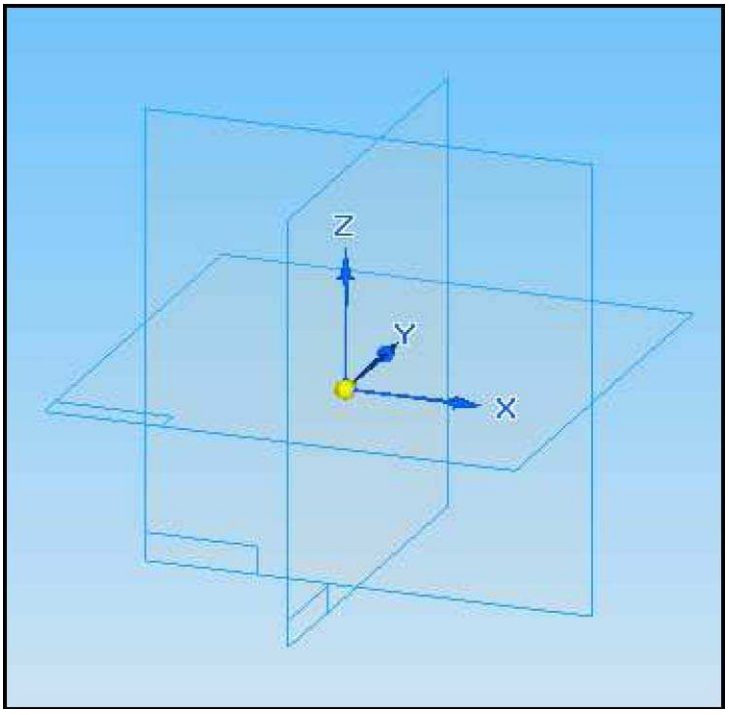


Fig. 3.23. Planos de referencia.

Existen tres Planos de Referencia predeterminados en Solid Edge. Se encuentran orientados en el espacio 3D relativos a las vistas de planta, alzado y perfil derecho.

3.3.4.3 Barra de menús

La barra de menús de Solid Edge contiene los mismos menús que la mayoría de las aplicaciones de Windows. Dentro de estos menús el programa visualiza un conjunto de comandos del entorno activo de Solid Edge figura 3.24.

Fig. 3.24. Barra de menús.

3.3.4.4 Cinta Smartstep

Se trata de una barra ubicada bajo la principal. Está asociada a los comandos, disponiendo cada uno de su propia cinta. Guían al usuario a través de la secuencia del comando, especificando en cada momento el paso en el cual se haya la operación, figura 3.25.

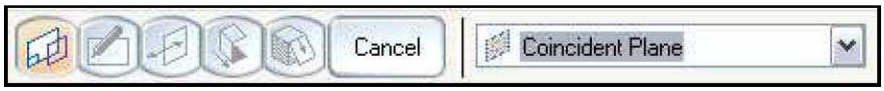


Fig. 3.25. Cinta Samartstep.

3.3.4.5 Barras de herramientas

PRINCIPAL: contiene los comandos comúnmente utilizados para manipular documentos, imprimir documentos y tratar vistas, figura 3.26.

Fig. 3.26. Barra de herramientas principal.

ESPECÍFICAS: contiene los principales comandos utilizados en el entorno. Se utilizarán estos comandos para construir modelos de pieza en este caso, figura 3.27.

Fig. 3.27. Barra de herramientas especifica.

3.3.4.6 EDGEBAR

El Edgebar es una ventana que por defecto se presenta en la parte izquierda del interfaz. Esta herramienta da acceso a una serie de opciones que difieren según el entorno en el cual se esté, figura 3.28.

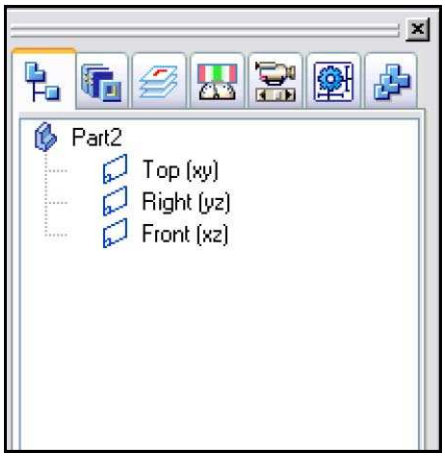


Fig. 3.28. Edgebar.

Una de las fichas del Edgebar mas utilizadas es Pathfinder de la figura 3.29

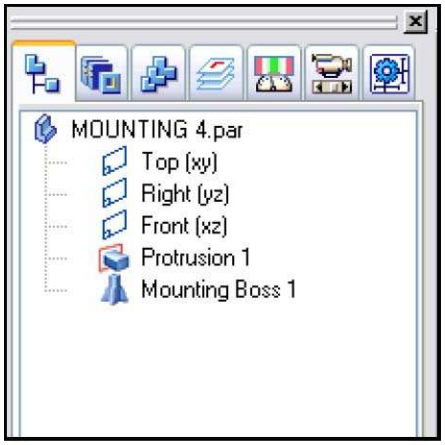


Fig. 3.29. Ficha de Edgebar , Pathfinder.

Cuya finalidad es guiar al usuario en el proceso de elaboración de la pieza, mostrando el conjunto de operaciones que la forman. De esta manera se tiene un rápido acceso a dichas operaciones para editarlas, visualizarlas, etc. Muestra las operaciones de construcción de una pieza de forma esquemática.

3.3.5. RobotStudio.

3.3.5.1 Introducción.

RobotStudio permite la programación fuera de línea y la simulación de los sistemas del robot en un PC Windows estándar. RobotStudio forma parte del controlador virtual ABB, que es una copia exacta del software real que se ejecuta en el robot de producción.

El controlador también contiene una unidad de programación virtual, que le permite manejar el robot simulado exactamente de la misma forma que un robot real.

Para empezar a trabajar con RobotStudio es necesario tener instalado los controladores de los robots llamados RobotWare.

3.3.5.2 Estación.

Para empezar a trabajar con RobotStudio, hay que crear una nueva estación con un robot. En este apartado aparecerá predefinido una ruta en la cual están todos los controladores RobotWare instalados, basta con seleccionar uno. Una vez seleccionado, arrancará el controlador virtual, si existen varias bibliotecas para éste lo indicará y basta con seleccionar la adecuada.

3.3.5.3 Interfaz.

La interfaz del usuario de RobotStudio consta de los siguientes elementos principales que se detallaran a continuación: Área gráfica, navegadores, ventana de salida, barra de estado, barras de herramientas y menús.

Cuando se inicia la aplicación se muestran los navegadores, el área gráfica y la ventana de salida, como se puede mostrar en la Fig 3.30.

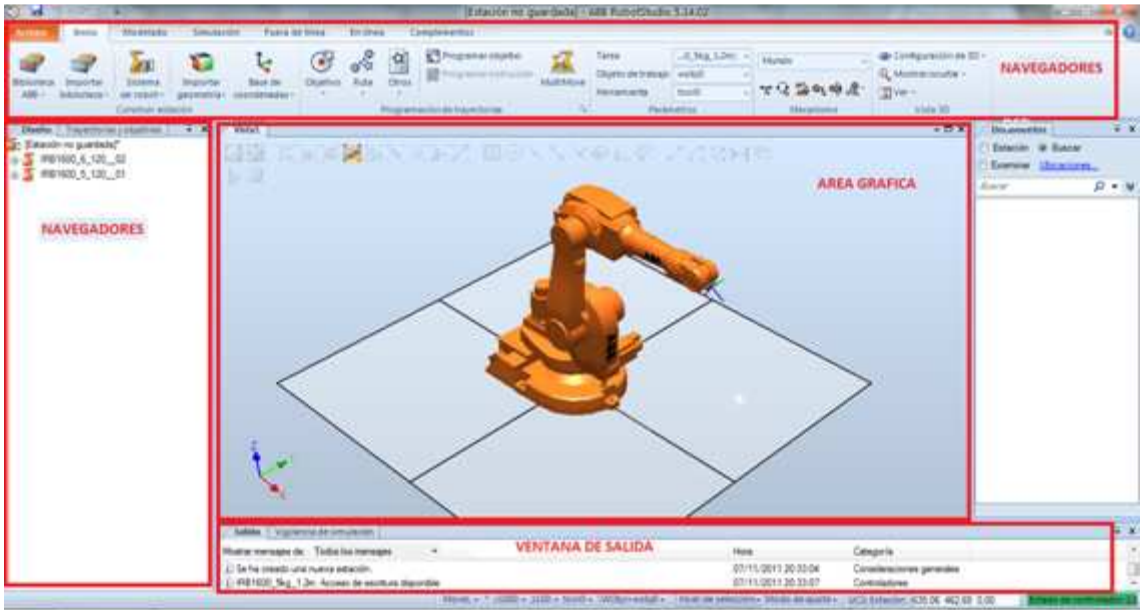


Fig. 3.30. Interfaz Robotstudio.

3.3.5.3.2 Área gráfica.

El área gráfica puede contener una o varias vistas gráficas que muestran su estación en 3D.

La vista gráfica es la visualización de la estación con sus objetos y simulaciones. En la vista gráfica, puede ver y manipular la representación en 3D de su estación.

3.3.5.3.3 La ventana de salida.

En la ventana de salida aparecen los mensajes dirigidos al usuario. La ventana de salida puede mostrar muchos tipos diferentes de mensajes y éstos se muestran en diferentes secciones en función del tipo de mensaje.

3.3.5.3.4 Navegadores.

Se pueden diferenciar tres grupos de navegadores:

- El navegador de propiedades.

El navegador de propiedades muestra todas las propiedades de los objetos seleccionados. En este navegador es posible cambiar las propiedades de los objetos como se puede ver en la figura 3.31. de forma general.

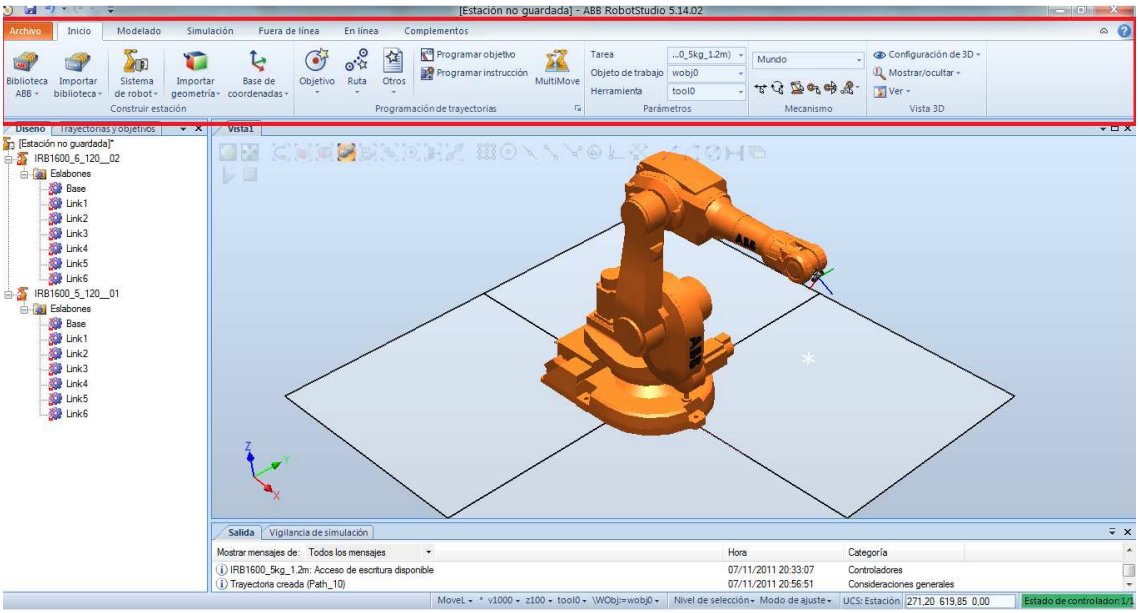


Fig. 3.31. Navegador de Propiedades General.

- El navegador de objetos.

El navegador de objetos muestra el contenido de la estación en una jerarquía textual. Si añade un objeto, el navegador inserta automáticamente el objeto en la estructura. Desde este navegador puede cambiar las propiedades de todos los objetos como se puede observar en la Fig 3.32.

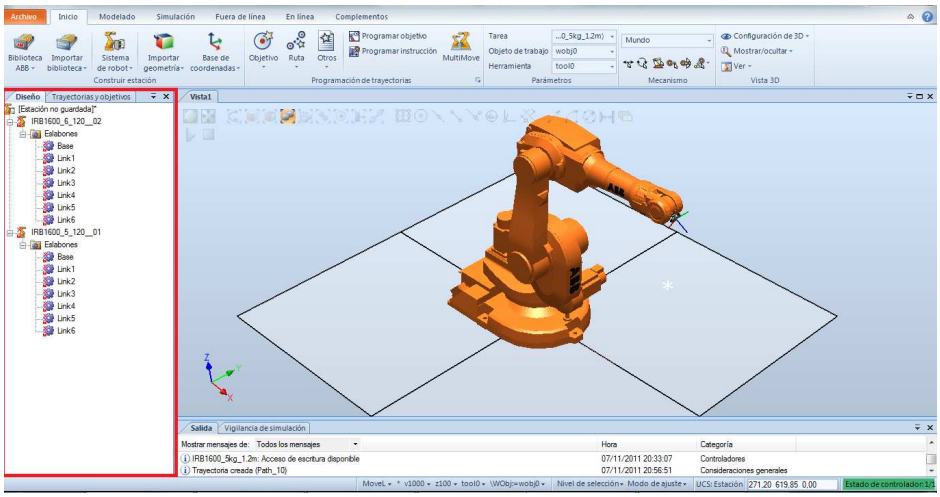


Fig. 3.32. Navegador de objetos.

Los objetos pueden ser componentes físicos, como robots y herramientas, o elementos lógicos, como Objetivos y Trayectorias.

- Navegador de programas.

El navegador de programas muestra las piezas estructurales del programa de RAPID, como se puede apreciar en la figura 3.33.

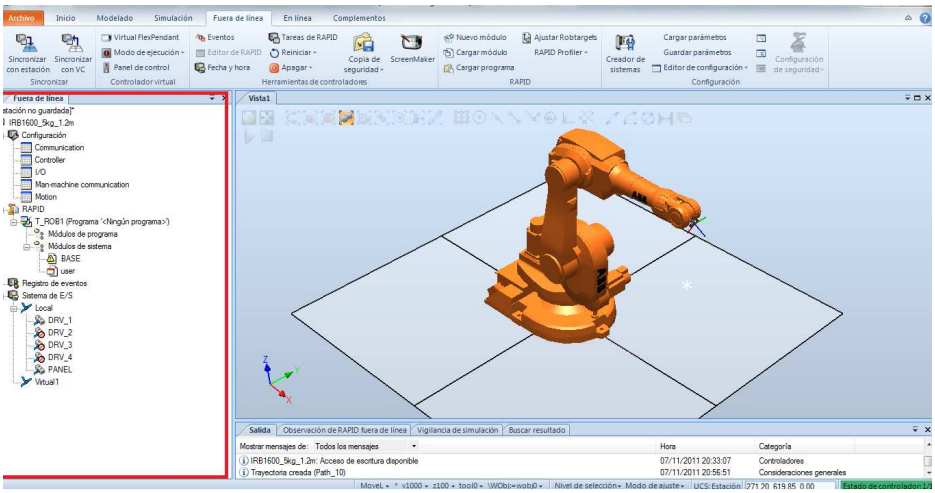


Fig. 3.33. Navegador de Programas.

Todas estas piezas de RAPID pueden crearse o eliminarse desde el navegador de programas.

Cada pieza de programa se representa mediante un nodo en el árbol

3.3.5.3.5 La barra de estado.

La barra de estado de la parte inferior de la ventana de RobotStudio permite obtener información breve y esencial sobre la estación.

En la figura 3.34 y 3.35 se puede observar dicha barra.

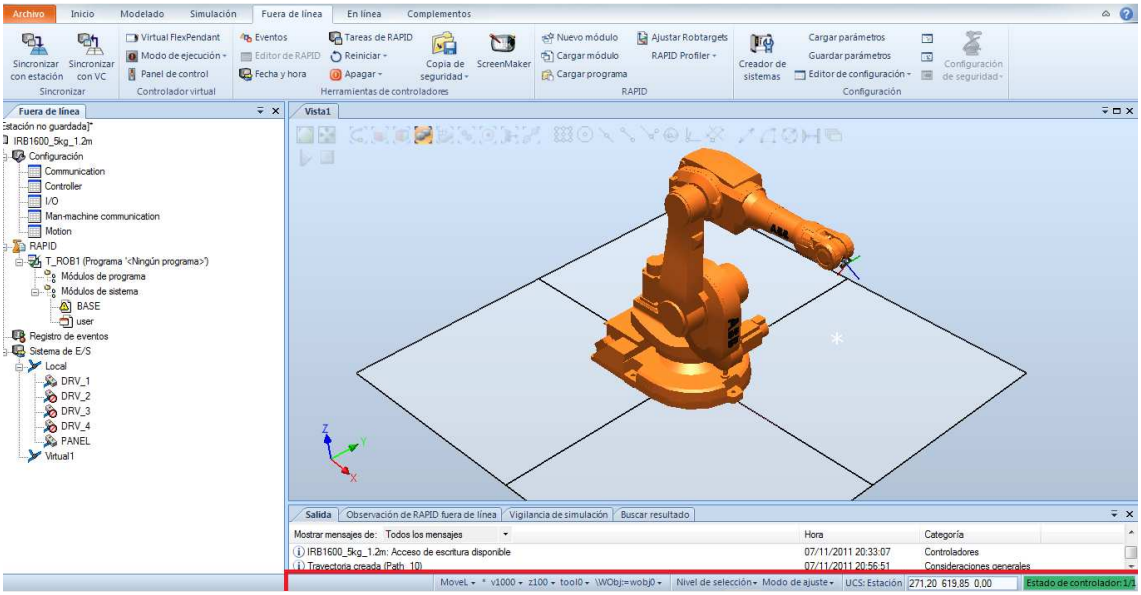


Fig. 3.34. Barra de Estado General.

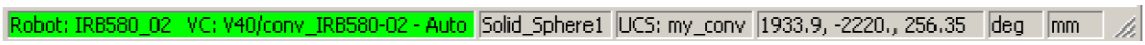


Fig. 3.35. Barra de Estado Detallado.

A continuación se describen los elementos de la barra, enumerados de izquierda a derecha como aparecen en la barra de estado de la ilustración anterior:

- **Robot activo.**

Muestra el nombre del robot activo y el controlador virtual. También muestra el modo de funcionamiento de los controladores virtuales.

Existen tres modos posibles: Automático, Manual y Parado

- **Objeto Seleccionado.**

Muestra el objeto que está seleccionado actualmente en la estación.

- **Sistema de coordenadas del usuario**

Muestra el objeto que está seleccionado actualmente en la estación.

- **Posición de la última selección**

Las coordenadas dadas son la posición de la última selección que haya realizado.

- **Unidad angular actual.**

Define qué unidad está utilizando para expresar los ángulos.

- **Unidad de longitud actual.**

Define qué unidad está utilizando para expresar las longitudes.

3.3.5.3.6 Barras de herramientas.

Para mantener un control total de la ventana de gráficos en todo momento durante el trabajo con RobotStudio, todas las ventanas de diálogo se muestran en la ventana de herramientas, que se muestra en la parte inferior del navegador objetos o en del navegador propiedades.

El contenido de la ventana de herramienta varía en función de qué ventana de diálogo esté activa. Sucesivamente se detallaran el contenido de las barras de herramientas más importantes.

3.3.5.3.7 Barra de herramientas Crear componente.

Puede usar la barra de herramientas Crear componente (Fig 3.36) cuando desee crear o modelar componentes en su estación.

Fig. 3.36. Barra Herramientas Crear Componente.

3.3.5.3.8 Barra de herramientas Simulación.

La barra de herramientas Simulación se utiliza para iniciar y detener la ejecución de sus simulaciones, es decir, la ejecución del programa de RAPID y está representada en la Fig 3.37.

Fig. 3.37. Barra Herramientas Simulación.

3.3.5.3.9 Barra de herramientas Nivel de selección.

En la barra de herramientas Nivel de selección representada en la Fig 3.38 se pueden escoger con qué tipo de objetos desea trabajar. Puede utilizar varios niveles de selección a la vez.

Fig. 3.38. Barra Herramientas Nivel Selección.

3.3.5.3.10 Barra de herramientas Ajustar a cuadrícula.

La barra de herramientas Ajustar a cuadrícula representada en la Fig 3.39, permite especificar el ajuste que se usará al elegir los objetos de la ventana gráfica.

Fig. 3.39. Barra Herramientas Ajustar Cuadricula

3.3.5.4 Menús.

El menú contextual ofrece una forma sencilla de obtener los comandos disponibles que pueden aplicarse a un objeto.

Mediante la selección de un objeto en el navegador de objetos o en la ventana gráfica con el botón derecho del ratón, se abre un pequeño menú contextual. En este menú sólo se muestran los comandos pertinentes para el objeto seleccionado.

En la Fig 3.40 queda definido dicho menú.

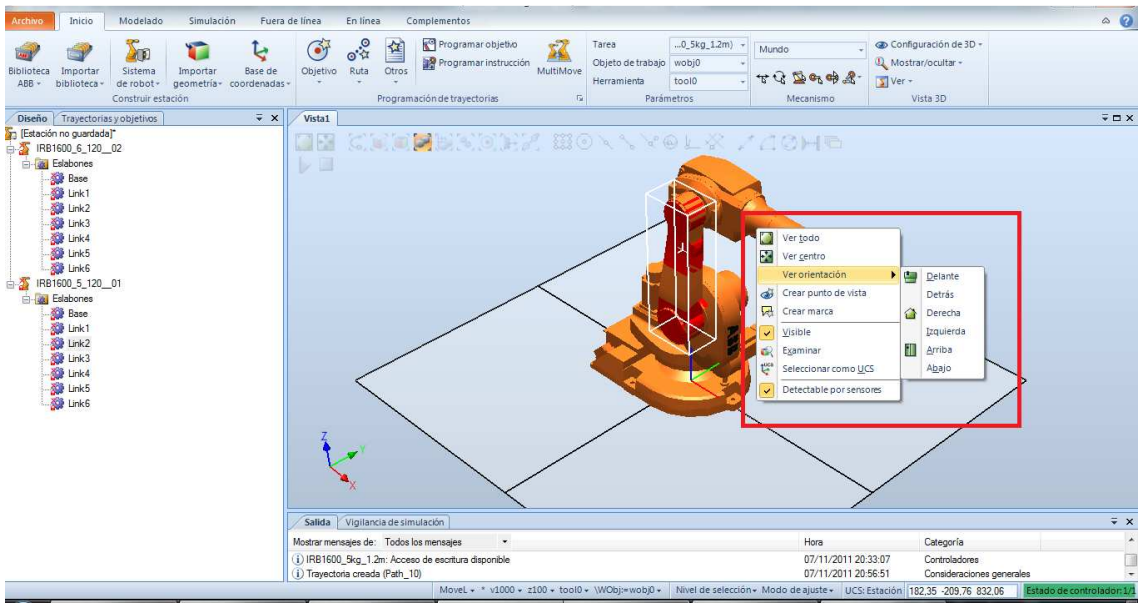


Fig. 3.40. Menú Contextual.

3.3.6. Software inicial de JR3.

Para el desarrollo del proyecto, se ha tomado como base el software "JR3 PCI & qPCI Interface Versión 3.0" el cual permite extraer y representar los valores de fuerza par de uno o varios sensores JR3. Además, este software también extrae del sensor valores tales como números de serie u otra información la cual para la aplicación a tratar se considera poco útil.

Debido a las numerosas funciones que ofrece el software inicial, de las cuales muchas de ellas en este trabajo no son necesarias, se procede a la modificación del código y la interfaz para adaptarlo a las necesidades del proyecto.

3.3.6.1 Desarrollo del funcionamiento.

Como el código fuente inicial ya se encuentra implementado y debida a su gran extensión únicamente se van a nombrar muy brevemente los diferentes archivos y cabeceras utilizadas.

Las partes que han sido diseñadas para este proyecto se explicaran posteriormente en capítulos sucesivos. Los archivos y cabeceras completos se pueden encontrar en los anexos del proyecto.

- **jr3pci.cpp.**
  Este es el archivo de la aplicación principal que contiene la clase CJr3pciApp.

- **ProgDlg.h y progdlg.ccp.**
  Estos archivos son los encargados de controlar la interfaz principal.

- **jr3pciDlg.h,jr3pciDlg.cpp.**
  Estos archivos contienen la clase CJr3pciDlg. Esta clase es la encargada de definir el comportamiento del cuadro de dialogo principal de la aplicación.

- **DlgProxy.h y DlgProxy.cpp.**
  Estos archivos contienen su clase CJr3pciDlgAutoProxy. Esta clase se llama el "proxy de automatización". Y se encarga de exponer los métodos automatización y propiedades que los controladores de automatización pueden utilizar para acceder a su diálogo.

- **StdAfx.h y StdAfx.cpp.**
  Estos archivos se utilizan para crear un encabezado precompilado (PCH) jr3pci.pch nombre y un archivo de tipos precompilado denominado StdAfx.obj.

- **jr3pci.h.**
  Este es el archivo de encabezado principal de la aplicación. y declara la aplicación clase CJr3pciApp.

- **Resource.h.**
  Este es el archivo de encabezado estándar que define los nuevos identificadores de recursos. Microsoft Visual Studio 2008 lee y actualiza este archivo.

3.3.6.2 Interfaz inicial JR3.

La interfaz que JR3 ofrece para la lectura de datos en sensores fuerza par, como se ve en la Fig 3.41, se trata de una interfaz la cual ofrece demasiados datos los cuales en este trabajo se ignoran.

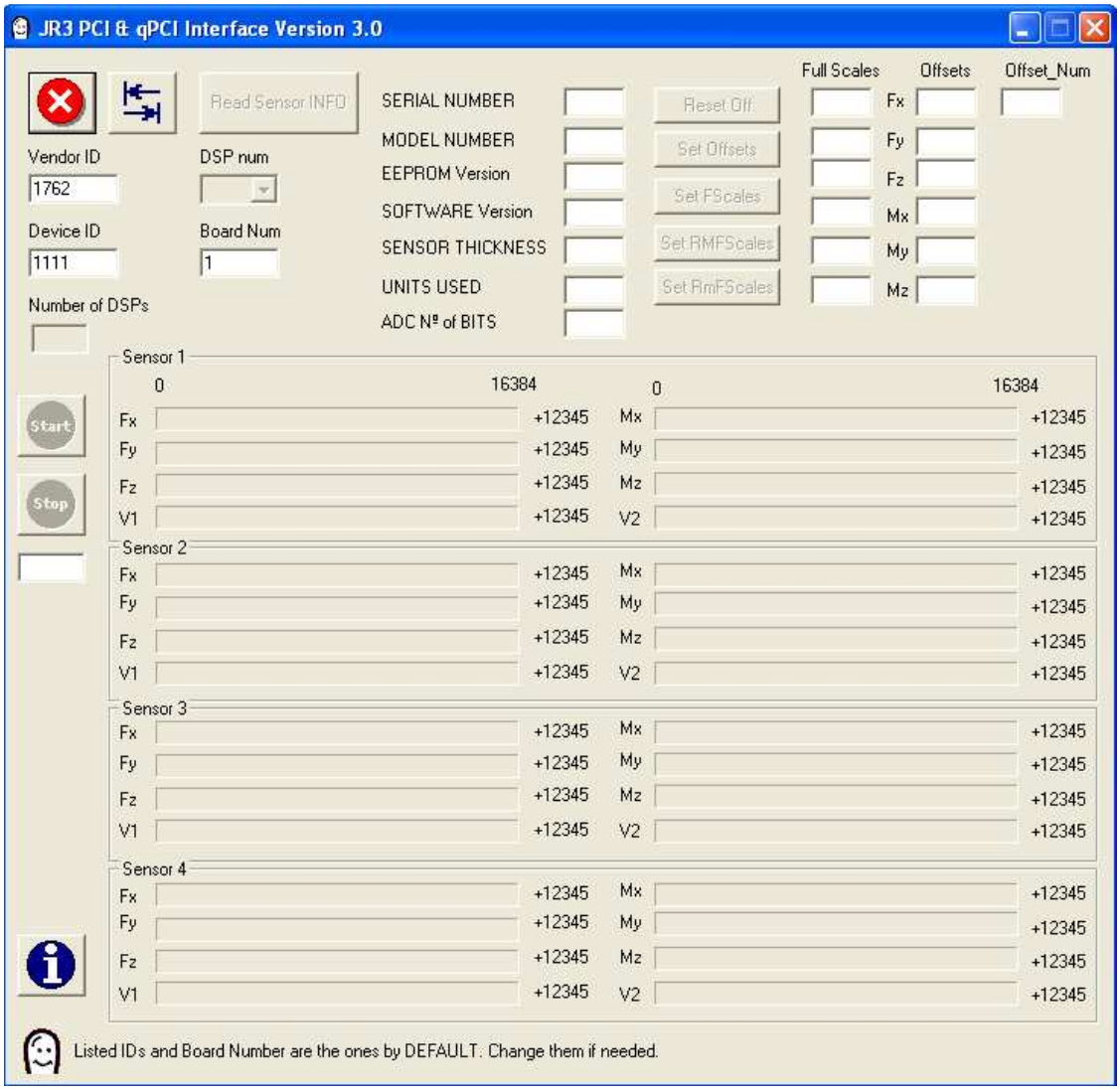


Fig. 3.41. Interfaz inicial de JR3.

Se describen a continuación las principales características de la aplicación inicial de JR3:

- Botones de conexión con el sensor y cierre de la interfaz. Mediante los botones representados en la Fig 3.42, el usuario podrá conectar el PC con el sensor JR3 siempre y cuando haya indicado correctamente el número de identificación de la tarjeta PCI encargada de la captura de los datos (Device ID).

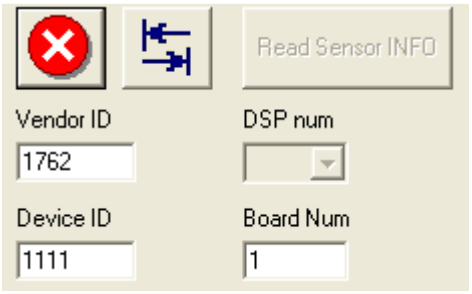


Fig. 3.42. Interfaz JR3 inicial – Botones de conexión y cierre.

- Botones de "Start" y "Stop". Representados en la Fig 3.43 estos botones permiten que una vez realizada la conexión del PC con el sensor JR3, la interfaz comience a representar los valores de fuerza y par extraídos.

Fig. 3.43. Interfaz JR3 – Botones "Start" y "Stop".

- Posibilidad de conectar hasta 4 sensores JR3 fuerza par y representar sus valores de manera independiente.

- Botón de "Read Sensor INFO" el cual permite al usuario conocer toda la información del sensor JR3 de fuerza par conectado. En la Fig 3.44 se observa la zona descrita de la interfaz.

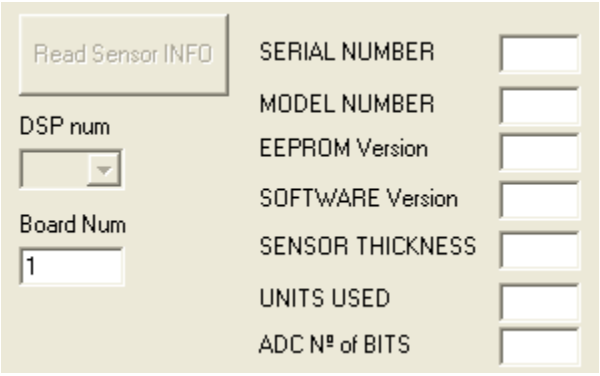


Fig. 3.44. Interfaz JR3 inicial - Read Sensor INFO.

Los datos del sensor representados son:

i. Número de serie.
ii. Número de modelo.
iii. Versión de memoria EEPROM.
iv. Versión del software.
v. Espesor.
vi. Unidades utilizadas.
vii. Número de bits del conversor analógico – digital.

- Reset de los offsets del sensor. Mediante el botón "Reset Off", los valores de offset representados en la tabla de Offsets correspondientes al sensor toman un valor 0 y de esta manera el sensor ignora las fuerzas y pares existentes justo en el momento de hacer click en dicho botón. Se observa en la Fig 3.45 la zona de la interfaz encargada del reset de los valores de offsets.

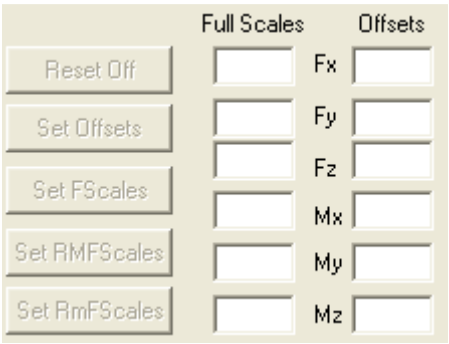


Fig. 3.45. Interfaz JR3 inicial – Reset y Set Offsets.

- Set de los offsets del sensor. Mediante el botón "Set Offsets" (véase Fig 3.45), la representación de los valores del sensor pasa a tener un offset con el valor indicado en la zona que representa el offset actual del sensor. Estos valores pueden ser modificados y pasan a aplicarse cuando se hace click en este botón.

- Cambio de escalas en los valores representados. Mediante los botones de la Fig 3.46, el programa inicial de JR3 permite al usuario la modificación de las escalas aplicadas en los valores leídos del sensor JR3.

Fig. 3.46. Interfaz JR3 inicial – Scales.

Mediante el botón "Set FScales", la interfaz permite aplicar a los valores representados en ese instante una escala la cual toma el valor indicado en la tabla "Full Scales" representada en la Fig 3.56. Estos valores pueden ser modificados por el usuario en cualquier momento y aplicados a la representación mediante un click en dicho botón.

Por otra parte, los botones "Set RMFScales" y "Set RmFScales" permiten al usuario aplicar escalas ya definidas y que solo pueden ser modificadas editando el código del programa.

- Zona de representación de los estados de la interfaz. La zona representada en la Fig 3.47, indica al usuario los distintos estados por los que pasa la interfaz como por ejemplo el inicio de lectura de datos o cambios en los valores de escalas y offsets.

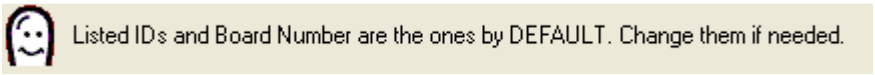


Fig. 3.47. Interfaz JR3 inicial – Estados de la interfaz.

- Botón de información de la interfaz. Representado en la Fig 3.48, este botón muestra al usuario la información sobre el creador de la interfaz y otros datos de interés.

Fig. 3.48. Interfaz JR3 inicial – Botón de INFO.

4. DESARROLLO DEL PROYECTO PARTE HARDWARE.

4.1. Introducción.

La parte hardware de este proyecto surge a partir de la necesidad de adaptar el sensor JR3 de fue fuerza-par al robot ABB.

La mejor ubicación para el sensor de fuerza-par JR3 una vez estudiado la forma del robot ABB y las necesidades de utilización del sensor, se dispone que su colocación sea en la última articulación, es decir, irá colocado en la punta del robot ABB, véase figura 4.1, para poder así captar la información de la fuerza que es ejercida por la persona sobre el sensor.

Fig. 4.1. Extremo robot ABB.

La figura 4.2 es un resumen, donde se explica cómo es el montaje, las partes de las que consta y la colocación de las piezas que lo forman.

Como se puede ver está formado por las dos placas que se diseñan, el sensor JR3, robot ABB y un soporte de empuje, siendo esta la pieza sobre la cual la persona realiza una fuerza en una dirección.

Fig. 4.2. Despiece instalación de sensor en robot ABB.

Para fijar el sensor al Robot ABB se implementan dos piezas metálicas, que mediante tornillos nos permiten instalar el sensor. En la figura siguiente 4.3 se muestra como quedaría el montaje con todas las piezas instaladas y con una carga ejerciendo la fuerza.

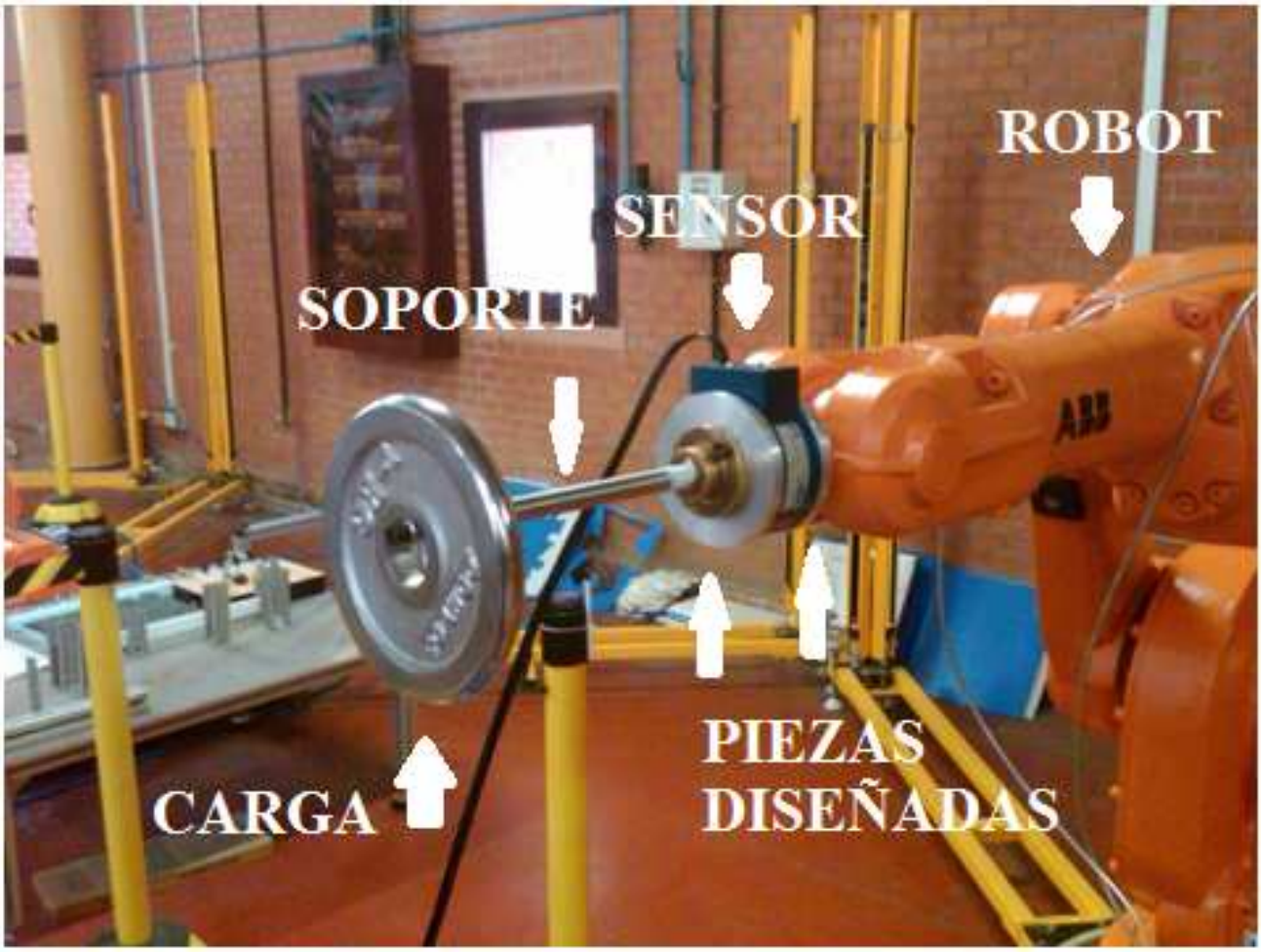


Fig. 4.3. Montaje final.

4.2. Diseño de piezas.

Las piezas a diseñar son dos y se explicaran a continuación.

4.2.1. Pieza de unión del robot con el sensor.

La primera placa servirá para unir la punta del Robot ABB al Sensor JR3 .En la siguiente figura 4.4 podemos observar dicha placa durante el modelado web.

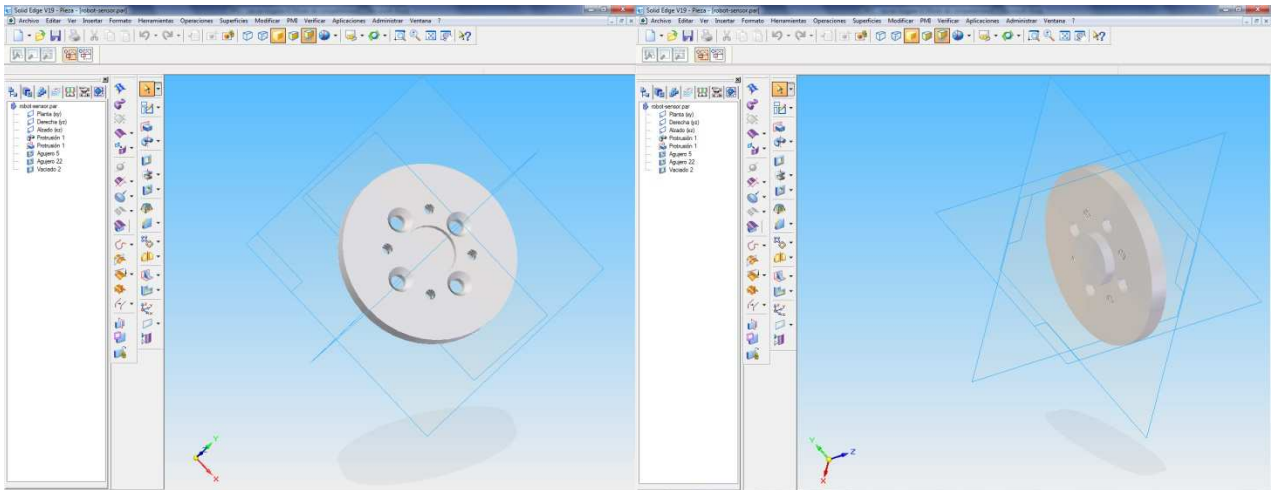


Fig. 4.4. Pieza robot-sensor durante el desarrollo web.

La pieza es un disco metálico de aluminio con un sistema de guiado para su colocación macho –hembra.

Dispone de cuatro taladros con chaflán grandes ,para sujetar por un lado el propio disco al robot y cuatro roscas más pequeñas de métrica M3 para fijar el sensor JR3 al disco.

En la figura 4.5 se muestra el resultado definitivo en aluminio.

Fig. 4.5. Pieza terminada robot-sensor.

En el anexo 12.4 se encuentra un plano con las medidas y dimensiones detalladas de la pieza.

4.2.2. Pieza de unión sensor adaptador

La segunda placa es utilizada para poder colocar un adaptador al sensor, con la finalidad de ser dicho adaptador, por donde la persona ejerza la fuerza que el sensor capta.

En la siguiente figura 4.6 se puede observar dicha placa durante el modelado web.

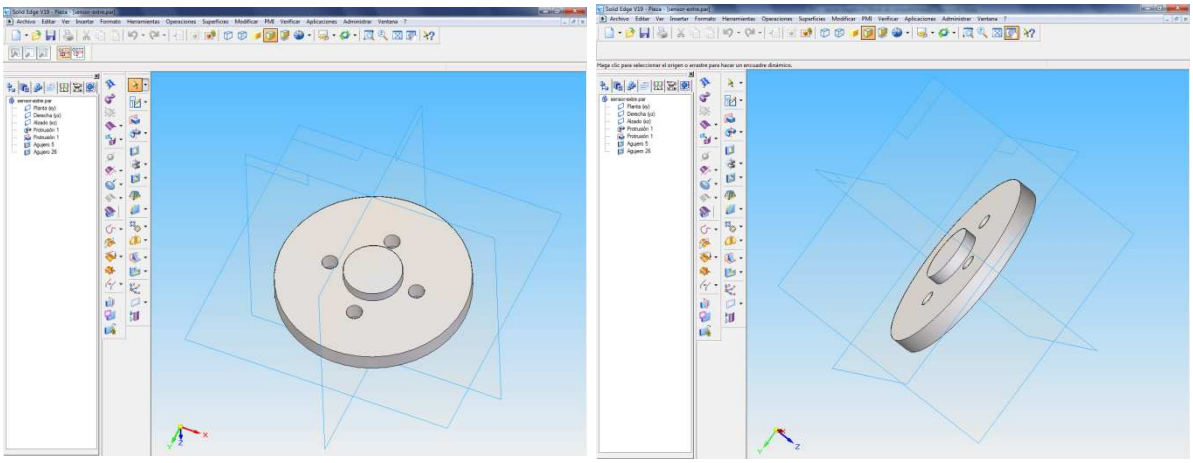


Fig. 4.6. Pieza sensor-robot durante el desarrollo web.

La pieza es un disco metálico de aluminio con un sistema de de guiado para su colocación macho –hembra; y cuatro taladros con chaflán para sujetar el soporte de empuje al sensor JR3.

En la figura 4.7se muestra el resultado definitivo en aluminio.

Fig. 4.7. Pieza terminada sensor-adaptador.

En el anexo 12.5 se encuentra un plano con las medidas y dimensiones detalladas de la pieza.

4.2.3. Sistema de fijación.

Para fijar estas placas se utiliza el tornillo:

- De métrica 7 y con cabeza avellanada a 65 grados y una longitud de rosca de 7 mm. Véase figura 4.8.

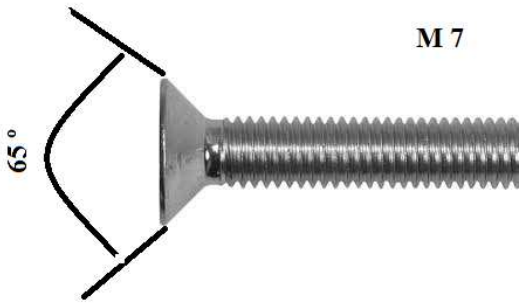


Fig. 4.8. Tornillo sujeción robot placa.

El propio sensor JR3 dispone de tornillos propios para su agarre de rosca M6 como la figura 4.9 y de 7 mm de longitud.

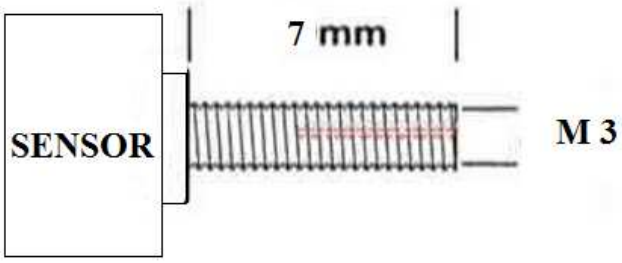


Fig. 4.9. Rosca sensor JR3.

5. DESARROLLO DEL PROYECTO PARTE SOFTWARE.

5.1. Introducción.

Para entender mejor el desarrollo de la aplicación implementada se muestra el siguiente diagrama de flujo. (Véase Fig 5.1).

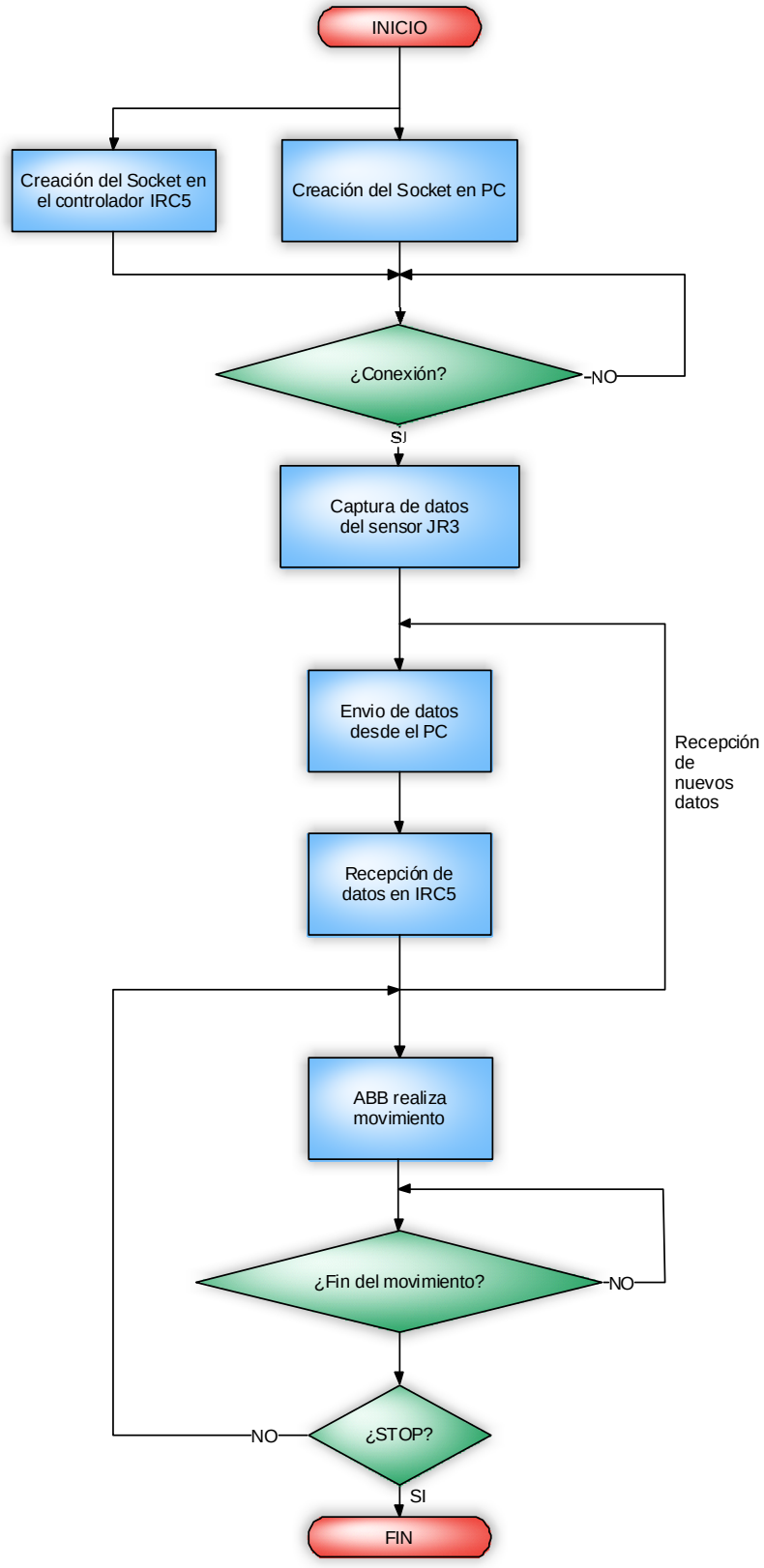


Fig. 5.1. Diagrama de flujo del sistema.

Para llevar a cabo la aplicación se trata de añadir al software proporcionado por JR3 para la interpretación de los datos obtenidos por la tarjeta PCI, una comunicación TCP/IP que envíe esos datos al controlador IRC5 mediante un cable Ethernet con una trama previamente confeccionada. En la Fig 5.2 se puede ver el tipo de comunicación entre cada uno de los bloques.

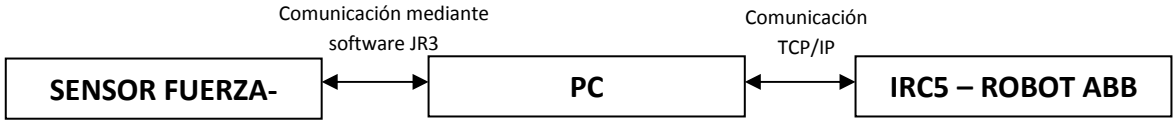


Fig. 5.2. Comunicación entre los distintos sistemas.

5.2. Desarrollo de la recepción y envío de datos desde el PC.

5.2.1. Funciones principales del nuevo software.

Las principales funciones del nuevo software obtenido mediante la modificación del proporcionado por JR3 son principalmente las siguientes:

- Comunicación y envío de datos desde el PC al robot ABB. Mediante esta característica, el nuevo software permite al usuario enviar los valores capturados por el sensor de fuerza par al robot ABB.

- Botones de conexión, start y stop. Botones con funciones similares a las del software original que gestionan el comienzo de la comunicación del sensor con el PC y la representación de los valores leídos.

- Representación de los valores de fuerza par. Al igual que la aplicación original de JR3, la nueva aplicación permite al usuario ver los valores de fuerza para ejercidos en el sensor.

- Almacenar datos de sensor JR3. La nueva aplicación permite al usuario almacenar los datos representados en la interfaz en un archivo .txt.

5.2.2. Desarrollo del funcionamiento.

Una vez se analizan los puntos que el nuevo software debe cumplir, mediante Microsoft Visual Studio 2008 se procede a modificar el programa.

5.2.2.1 Sockets.

Antes poder modificar el código se necesitaba realizar una conexión del PC con el robot. El método utilizado es el de los sockets [10].

Socket designa un concepto abstracto por el cual dos programas (posiblemente situados en computadoras distintas) pueden intercambiar cualquier flujo de datos, generalmente de manera fiable y ordenada

Para que dos programas puedan comunicarse entre sí es necesario que se cumplan ciertos requisitos:

- Que un programa sea capaz de localizar al otro.
- Que ambos programas sean capaces de intercambiarse cualquier secuencia de octetos, es decir, datos relevantes a su finalidad.

Para ello son necesarios los tres recursos que originan el concepto de *socket*:

- Un protocolo de comunicaciones, que permite el intercambio de octetos.
- Un par de direcciones del protocolo de red (dirección IP, si se utiliza el protocolo TCP/IP), que identifican la computadora de origen y la remota.
- Un par de números de puerto, que identifican a un programa dentro de cada computadora.

Los *sockets* permiten implementar una arquitectura cliente-servidor. La comunicación debe ser iniciada por uno de los programas que se denomina programa "cliente". El segundo programa espera a que otro inicie la comunicación, por este motivo se denomina programa "servidor".

Un *socket* es un proceso o hilo existente en la máquina cliente y en la máquina servidora, que sirve en última instancia para que el programa servidor y el cliente lean y escriban la información. Esta información será la transmitida por las diferentes capas de red.

Las propiedades de un *socket* dependen de las características del protocolo en el que se implementan. En este proyecto el utilizado es Transmission Control Protocol; una alternativa común a éste esUser Datagram Protocol.

Cuando se implementan con el protocolo TCP, los *sockets* tienen las siguientes propiedades:

- Son orientados a la conexión.
- Se garantiza la transmisión de todos los octetos sin errores ni omisiones.

- Se garantiza que todo octeto llegará a su destino en el mismo orden en que se ha transmitido.

Estas propiedades son muy importantes para garantizar la corrección de los programas que tratan la información.

El protocolo UDP es un protocolo *no* orientado a la conexión. Sólo se garantiza que si un mensaje llega, llegue bien. En ningún caso se garantiza que llegue o que lleguen todos los mensajes en el mismo orden que se mandaron. Esto lo hace adecuado para el envío de mensajes frecuentes pero no demasiado importantes, como por ejemplo, mensajes para las actualizaciones de un gráfico.

Para facilitar trabajo se han usado un conjunto de librerías que facilitan el desarrollo de las aplicaciones escritas en C++ que requieran comunicaciones vía TCP.

Están librerías son llamadas SolarSokets. Y están formadas por una serie de funciones que se pueden aplicar para establecer la conexión.

Como ya se ha dicho anteriormente el protocolo está formado por un servidor y un cliente , como se puede ver en la Fig 6.1 .El cliente se va a desarrollar en C++ mientras que el servidor se desarrollará en Rapid con los sokets que ABB implementa y que más adelante se explicaran.

SolarSokets contiene muchas funciones, pero para este proyecto solo se utilizan las siguientes:

- bool Connect (string Server, int Port);

Ordena al socket conectarse a la IP o URL Server por el puerto Port.

- bool Close ();

Termina la conexión con el servidor.

- bool Send (string Data);

Ordena enviar la cadena Datos, la cual es una cadena de caracteres ASCII.

- bool Send (const char *Data, unsigned int Length);

Ordena enviar una cadena de chars contenida en Data, de longitud Length, a diferencia de la anterior esta si se puede usar para enviar información binaria.

- void OnConnect (void (*_OnConnect)(void));

Recibe como parámetro una función que será llamada cuando ocurra la conexión del Socket.

Más adelante se procede a identificar de forma detallada el uso de dichas funciones de forma más especifica. En la Fig 5.3 se puede ver el diagrama de funcionamiento del protocolo de conexión utilizado.

- Primero el robot abre la comunicación y se mantiene a la espera de recepción de datos (1).
- Ahora el ordenador se conecta el robot y le empieza a mandar los datos (2 y 3).
- Con los datos enviados el robot realiza el movimiento (4).
- Y ese movimiento se realiza mientras el ordenador sigua mandando datos (3 y 4).
- Cuando se desea cortar la conexión lo único que se ha de hacer es pulsar el botón de “stop” en el programa JR3 del ordenador (5).

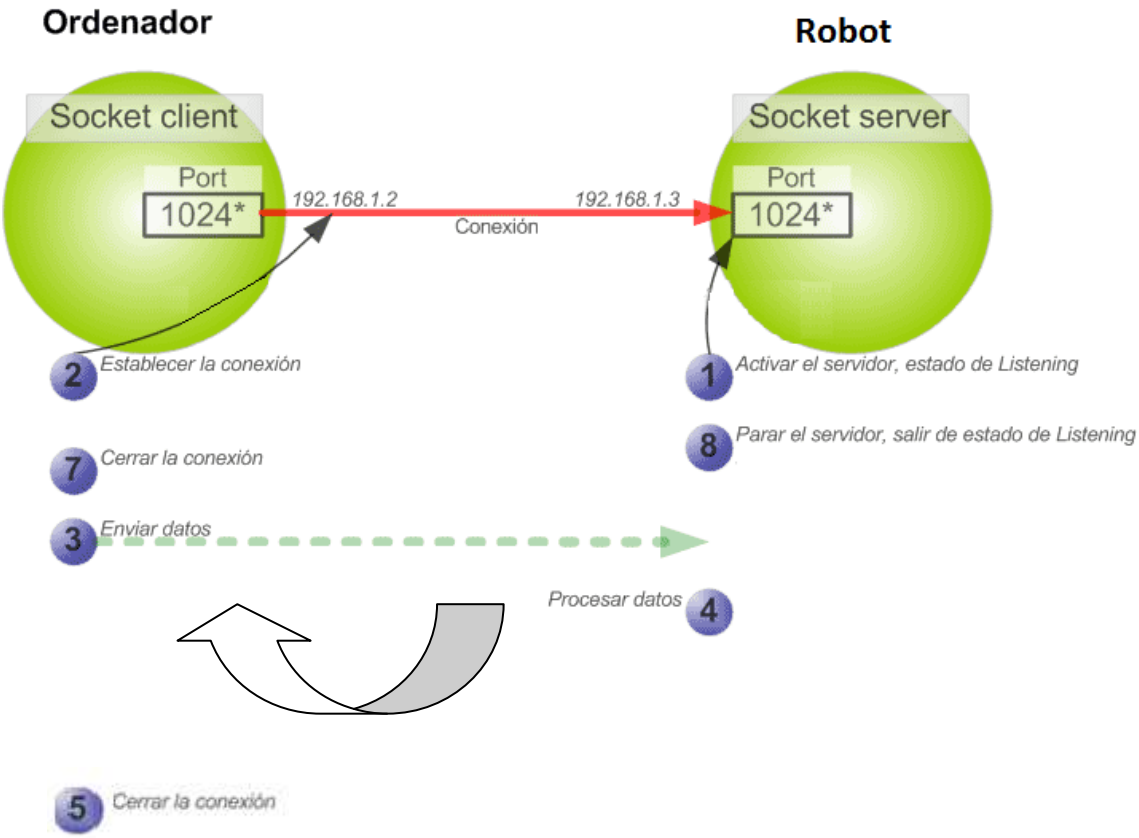


Fig. 5.3. Diagrama de funcionamiento del protocolo de conexión.

5.2.3. Nuevas declaraciones y creación de fichero txt.

Debido a que el software proporcionado por JR3 está diseñado únicamente para extraer y representar los valores obtenidos por el sensor, para adaptarse a la nueva tarea es necesario extraer los valores que el sensor va leyendo para enviárselos al robot ABB. Además, ya que la nueva aplicación permite al usuario almacenar los valores leídos por el sensor, se procede también a la creación del archivo txt encargado de almacenarlos.

En el código 1 se declaran todas las variables implementadas.

En el código se observan cómo se declaran todos los strings que son cadenas de caracteres donde se almacenan los valores de las fuerza ejercida por la persona

Las demás variables son de tipo entero usadas en los contadores y la comunicación así como los límites inferiores y superiores de las fuerzas, etc.

Con las variables "char" definimos una cadena de caracteres que albergara el numero de bits de los que está formada la fuerza en cada momento

También mediante "FILE*pFile=fopen("datos_sensor.txt")" se genera el txt para almacenar los datos si el usuario lo indica.

```
//Declaraciones de las variables necesarias para la nueva aplicación.

string datofx="",datofy="",datofz="",datomx="",datomy="",datomz="";
string stringlongitudfx="",stringlongitudfy="",stringlongitudfz="";
string datocompleto="";
char charlongitudfx[1],charlongitudfy[1],charlongitudfz[1];
short primeravez=1;
int contador=0;
int nTempx=0;
int nTempy=0;
int nTempz=0;
int longitudfx=0;
int longitudfy=0;
int longitudfz=0;
int maximo=13000;
int minimo=150;
int almacenardatos=0;


//Creación del fichero para almacenar los datos leidos.
FILE * pFile= fopen ("datos_sensor.txt","w");

  ……

  ……
```

Código 1: Nuevas declaraciones y creación de fichero txt.

5.2.4. Conexión con el PC.

Esta parte del código es la encargada de realizar la conexión del PC con el robot ABB. Mediante este proceso de conexión que se ejecuta al indicar al PC que se conecte, se procede a comenzar la comunicación con la IP y el puerto que se indican.

También, en esta parte del código que se muestra a continuación se crean las columnas donde se almacenaran los valores de fuerza y par leídos por el sensor para una mayor comodidad del usuario.

Mediante "fputs" se escribe por primera vez el fichero txt para guardar los datos si el usuario lo indica. Para crear una tabla organizada se escriben las columnas de fuerzas y momentos para después guardar los datos debajo.

Se observa como mediante "Cliente.Connect(192.168.133.2",4410)" se indica al PC la IP del controlador del robot así como el puerto que se va a utilizar para la conexión.

```
……
……
//Creación de las columnas en el fichero para almacenar los datos guardados.
        fputs (" Fx      Fy      Fz      Mx      My      Mz",pFile);
        fputs ("\n",pFile);
//Conexíón del PC con el robot ABB. Espera de 100 ms para establecer la conexión correctamente.
Cliente.Connect("192.168.133.2", 4410);
……
……
```

Código 2: Conexión con el PC.

5.2.5. Extracción de los valores leídos por el sensor.

Debido a la característica del programa original de JR3 de únicamente representar los valores leídos por el sensor, es necesario el modificar en esta nueva aplicación el código y almacenar en una variable, que formará la trama a enviar, los valores de fuerza que el sensor lee.

En el código 3 ya se encuentra en la aplicación inicial de JR3, y simplemente se añade el comando "atoi" que nos permite extraer el valor de la fuerza y copiarlo en la variable temporal "nTempx" de carácter entero para tratarla después y hacer la trama que se va a enviar.

La necesidad de usar este comando viene impuesto por el formato inicial del valor de la fuerza .Este formato es una cadena de caracteres y resulta extremadamente difícil poder trabajar con ello; de ahí, que se tuviera que cambiar a un valor entero.

Este código se repite otras cinco veces más una por cada fuerza y par del eje x,y,z

```
//Estracción de los valores de fuerza. Conversión mediante atoi a un entero para su comparación.

    msg.Format("%d",off.fx);
    m_fx.SetWindowText(msg);
    nTempx = atoi (msg);
  ……
  ……
```

Código 3: Extracción de los valores leídos por el sensor.

5.2.6. Almacenar datos en fichero txt.

Una vez que los tres valores de fuerza y par están almacenados en una variable, si el usuario ha seleccionado la opción de almacenar los datos, las 6 variables pasan a incluirse en una nueva fila del fichero txt creado (código 4).

Mediante “fputs” se indica que cada vez que escriba los valores realice un salto de línea para crear asi la tabla. Con el comando “fwrite” se van escribiendo cada uno de los datos que se quieren registrar en el txt.

```
if (almacenardatos==1)
{
        fputs ("\n",pFile);
        fwrite (&datofx,sizeof(datofx),1,pFile);
  ……
  ……
```

Código 4: Almacenar datos en fichero txt.

5.2.7. Rango de medidas a enviar.

Debido a que el rango de medidas obtenidas por el sensor es muy amplio y que los movimientos que realiza el robot son proporcionales a la fuerza ejercida sobres su herramienta se procede a crear un rango entre ±150 y ±13000 para de este modo ignorar los valores pequeños que producen movimientos cortos cuando el robot está en reposo y valores muy grandes que sacan de rango al robot ABB. De esta manera para valores de menos de 150 en alguno de los ejes el PC enviará un valor de 0 en dicho eje mientras que para valores superiores a 13000 en algún eje el PC enviará un valor de 13000 y no superior. En el código 6 se trata el proceso descrito.

Simplemente en esta parte de código se comprueba que las variables temporales de fuerza (nTempx, nTempy y nTempz) estén dentro de un rango.

Primero se examina el límite inferior, usando la sentencia condicional llamada "if". Dentro de "IF" se comparan valores tanto positivos como negativos. En el caso que no se supere dicho límite el valor de las variables temporales pasan a ser 0.

Después, se miran los limites superiores para enviar el valor de 13000 (positivo o negativo según corresponda) para evitar movimientos muy grandes en el robot. Del mismo modo que se ve en el código tratar el eje x se tratarían los demás ejes.

```
if ((nTempx>-minimo) && (nTempx<minimo) && (nTempy>-minimo) &&
(nTempy<minimo) && (nTempz>-minimo) && (nTempz<minimo))
    {
        datofx="";
        datofy="";
        datofz="";
        nTempx=0;
        nTempy=0;
        nTempz=0;
    }
 if (nTempx<-maximo)
    {
        datofx="-13000";
    }
else if (nTempx>maximo)
    {
        datofx="13000";
     }
  ……
  ……
```

Código 5: Rango de medidas a enviar.

5.2.8. Eliminación de ruido producido por el sensor.

Cuando se produce la conexión entre el sensor JR3 y el PC, el sensor crea unas señales erróneas representando en el PC valores de fuerza importantes cuando realmente el sensor se encuentra en reposo. Si estos valores no fueran eliminados el robot recibiría datos erróneos y realizaría movimientos amplios cuando realmente debería estar parado. Este problema se genera también en la propia interfaz de JR3. En el código 6 se muestra como se trata este error.

El código 6 está formado por un bucle "for" que es realizado 4 veces asignando valores nulos los 4 primeros valores leídos hasta que el sensor se estabiliza y comienza a leer valores correctos. Simplemente se limpian los strings para no enviar nada y que el robot permanezca quieto hasta comenzar a leer las fuerzas.

Usando Cliente.Send(datocompleto) enviamos al robot el valor de la fuerza ,esperando previamente 500 milisegundos para que de tiempo a procesar los datos al robot .Este comando se explicara mas adelante.

```
if (primeravez==1)
{
        for (contador=0;contador<=4;contador++)
        {

    datocompleto="";
            stringlongitudfx="";
                    stringlongitudfy="";
            stringlongitudfz="";
                    datofx = "";
                    datofy = "";
                    datofz = "";
                    Sleep(500);
            Cliente.Send(datocompleto);
    primeravez=0;

        }
}
```

Código 6: Eliminación de ruido producido por el sensor.

5.2.9. Creación de la trama a enviar.

Una vez que todos los valores de fuerza a enviar se encuentran almacenados en distintas variables, se procede entonces a conocer la longitud de dichas variables para formar la trama. Tras conocer las 3 longitudes y las 3 fuerzas, concatenando estas variables se forma la trama como se puede ver en la Fig 4.5.

Longitud Fx	Longitud Fy	Longitud Fz	Fuerza en x	Fuerza en y	Fuerza en z

Fig. 5.4. Trama de datos enviada.

En el código 7 se observa cómo se crea la trama para la fuerza X (las demás fuerzas se tratan de la misma manera).

Primero se mide la logitud del dato mediante el comando “strlen” y se copia en la variable “logitudfx”. Se concatenan las 3 longitudes mediante “datocompleto+=….”. Después de igual forma se concatenan los valores de fuerza (datofx, datofy, datofz) para crear la trama completa y enviarle asi al robot un único string con toda la información que necesita para realizar el movimiento.

```
longitudfx=strlen(datofx.c_str());
  ……
  ……
datocompleto+=stringlongitudfx;
  ……
  ……
//Se completa la trama uniendo todos los valores .
datocompleto+=datofx;
datocompleto+=datofy;
datocompleto+=datofz;
```

Código 7: Creación de la trama a enviar.

5.2.10. Envío de la trama.

En el código 8 se puede ver el proceso encargado de enviar la trama creada. Después de enviar dicha trama se procede a limpiar todos los strings para no tener problemas en los siguientes datos que se envíen.

Lo primero que se observa es el comando “sleep”. Este comando se utiliza debido a la diferencia de velocidad de los procesadores del PC y del robot ABB ,procediendo así a ralentizar los envíos para evitar que el buffer del robot se sature demasiado.

Mediante "Cliente.Send(datocompleto)" se procede a enviar la trama completa creada descrita en el apartado anterior.

La demás parte de código se limita a "limpiar" las variables utilizadas.

```
Sleep(500);
Cliente.Send(datocompleto);
        datocompleto="";
        stringlongitudfx="";
    stringlongitudfy="";
        stringlongitudfz="";
        datofx = "0";
        datofy = "0";
    datofz = "0";
        longitudfx=0;
        longitudfy=0;
        longitudfz=0;
```

Código 8: Envío de la trama.

5.2.11. Inicio del envío de datos.

En este apartado se trata la orden que se genera para comenzar a enviar los datos al robot. Esta parte comprueba que la conexión se ha realizado correctamente y que se pueden comenzar a enviar los datos sin problema En el código 9 se trata el proceso.

```
void ClienteTest::onConnect()
{
   primeravez=1;
 }
```

Código 9: Inicio del envío de datos.

Mediante *onConnect* se da a conocer el momento que indica que la conexión con el robot se ha realizado correctamente y se procede a indicar que se van a empezar a enviar datos. Mediante la variable *primeravez* se indican que son los primeros datos y que deben ser tratados para evitar los errores descritos en el apartado 5.2.8 .

5.3. Desarrollo de la recepción e interpretación de los datos desde el robot ABB.

Para la realización del software encargado de recibir los datos leídos por el sensor JR3 y realizar así los movimientos oportunos teniendo en cuenta dichos valores, se ha utilizado un lenguaje en RAPID programado en ROBOT STUDIO.

La función principal del programa es poder crear una conexión con el sensor para así recibir los datos de este, procesarlos y por ultimo generar el movimiento del robot.

El código completo si se desea ver se encuentra en los anexos apartado 11.3

La explicación detallada del código se puede encontrar en la parte I de la memoria del proyecto.

5.4. Modificaciones la nueva interfaz para la aplicación.

5.4.1. Funciones principales del nuevo software.

Las principales funciones del nuevo software obtenido mediante la modificación del proporcionado por JR3 son principalmente las siguientes:

- Comunicación y envío de datos desde el PC al robot ABB. Mediante esta característica, el nuevo software permite al usuario enviar los valores capturados por el sensor de fuerza par al robot ABB.

- Botones de conexión, start y stop. Botones con funciones similares a las del software original que gestionan el comienzo de la comunicación del sensor con el PC y la representación de los valores leídos.

- Representación de los valores de fuerza y par. Al igual que la aplicación original de JR3, la nueva aplicación permite al usuario ver los valores de fuerza para ejercidos en el sensor.

- Almacenar datos de sensor JR3. La nueva aplicación permite al usuario almacenar los datos representados en la interfaz en un archivo .txt.

5.4.2. Descripción del interfaz

Esta nueva interfaz se ha reducido, como se ha comentado anteriormente, y adaptado para que cumpla de la forma más eficaz las especificaciones que el proyecto exigía.

En la figura 5.5 siguiente se puede ver la interfaz adaptada.

Fig. 5.5. Interfaz Sensor JR3 Adaptada

Se describen a continuación las principales características del interfaz adaptado de JR3:

- El primer paso para poder usar el programa es conectar la aplicación con la tarjeta de recepción de datos y para ello se tienen que introducir unos valores. Estos se encuentran definidos en la figura 5.6 y hacen referencia a la identificación del fabricante y del dispositivo una vez instalado en el bus de conexiones de la placa base.

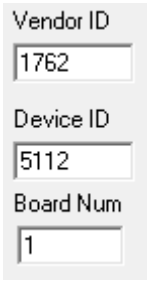


Fig. 5.6. Datos de conexiones.

- Una vez introducidos los datos se tiene que presionar el botón de conectar el programa con la tarjeta de recepción de datos. Este botón se encuentra en la figura 5.7.

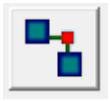


Fig. 5.7. Botón de conectar

- Una vez conectado con la tarjeta para visualizar los datos que el sensor está enviando se encuentra el botón de Start figura 5.8.

Fig. 5.8. Botón Visualizar valores.

- Estos valores de fuerza y par se representan como se puede ver en la figura 5.9 con un rango comprendido entre – 16384 y + 16384.esto valores se ordendan en dos columnas la de la derecha para la fuerza y la izquierda para el par. Ambos valores son enviados por el sensor en función de los tres ejes cartesianos X,Y,Z.

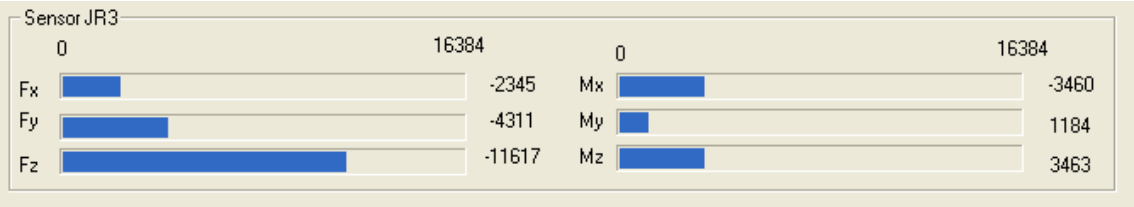


Fig. 5.9. Valores medidos por el sensor.

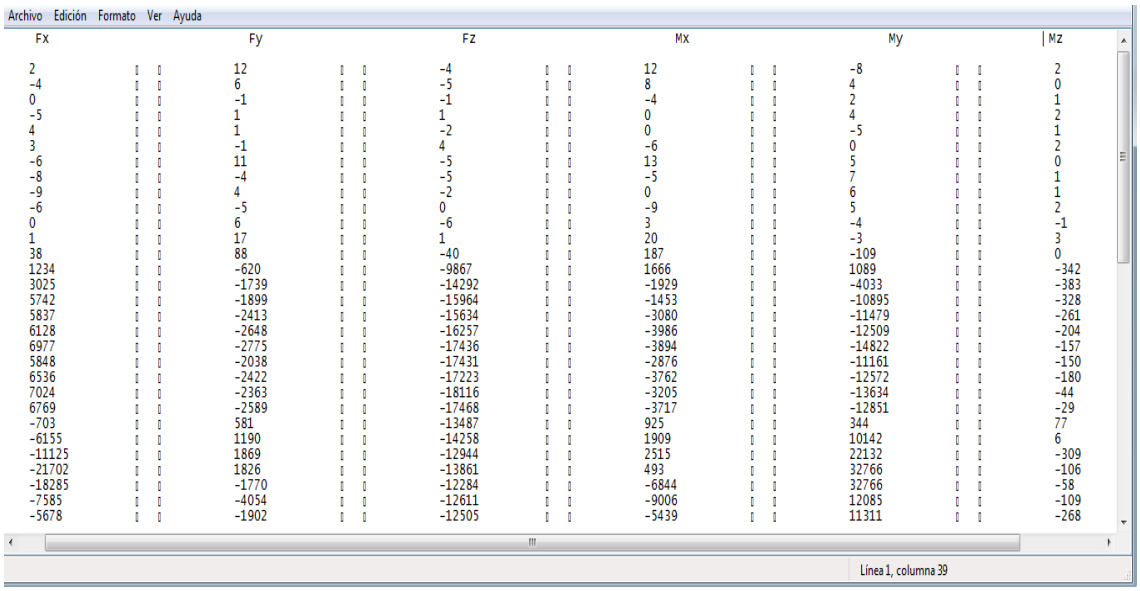
- Si se quiere parar de visualizar los valores y mantenerlos fijos en pantalla se encuentra el botón de stop representado en la figura 5.10 y con el que se puede parar la visualización.

Fig. 5.10. Botón de stop para la visualización de datos.

- Por último el botón ALMACENAR DATOS JR3 representado en la figura 5.11 es el encargado de generar un archivo txt que guarde los datos leídos por el sensor.

Fig. 5.11. Botón almacenar datos JR3.

Dichos valores se representan en 6 columnas; fuerza del eje x , fuerza del eje y, fuerza del eje z, par en eje x, par en eje y, par en eje z; de derecha a izquierda respectivamente. En la figura 5.12 se puede observar un ejemplo del archivo txt generado.



Archivo Edición Formato Ver Ayuda

Fx	Fy	Fz	Mx	My	Mz
2	12	-4	12	-8	2
-4	6	-5	8	4	0
0	-1	-1	-4	2	1
-5	1	1	0	4	2
4	1	-2	0	-5	1
3	-1	4	-6	0	2
-6	11	-5	13	5	0
-8	-4	-5	-5	7	1
-9	4	-2	0	6	1
-6	-5	0	-9	5	2
0	6	-6	3	-4	-1
1	17	1	20	-3	3
38	88	-40	187	-109	0
1234	-620	-9867	1666	1089	-342
3025	-1739	-14292	-1929	-4033	-383
5742	-1899	-15964	-1453	-10895	-328
5837	-2413	-15634	-3080	-11479	-261
6128	-2648	-16257	-3986	-12509	-204
6977	-2775	-17436	-3894	-14822	-157
5848	-2038	-17431	-2876	-11161	-150
6536	-2422	-17223	-3762	-12572	-180
7024	-2363	-18116	-3205	-13634	-44
6769	-2589	-17468	-3717	-12851	-29
-703	581	-13487	925	344	77
-6155	1190	-14258	1909	10142	6
-11125	1869	-12944	2515	22132	-309
-21702	1826	-13861	493	32766	-106
-18285	-1770	-12284	-6844	32766	-58
-7585	-4054	-12611	-9006	12085	-109
-5678	-1902	-12505	-5439	11311	-268

Línea 1, columna 39

Fig. 5.12. Archivo .txt generado con los valores del sensor.

6. INSTRUCCIONES DE PUESTA EN MARCHA.

En este apartado se definirán los procedimientos que se tienen que seguir para poder controlar el robot mediante el sensor JR3.

6.1. Puesta en marcha hardware.

6.1.1. Conexión del hardware.

- El primer paso es colocar el sensor JR3 en la punta del robot y conectarlo al ordenador como se muestra en la 6.1.

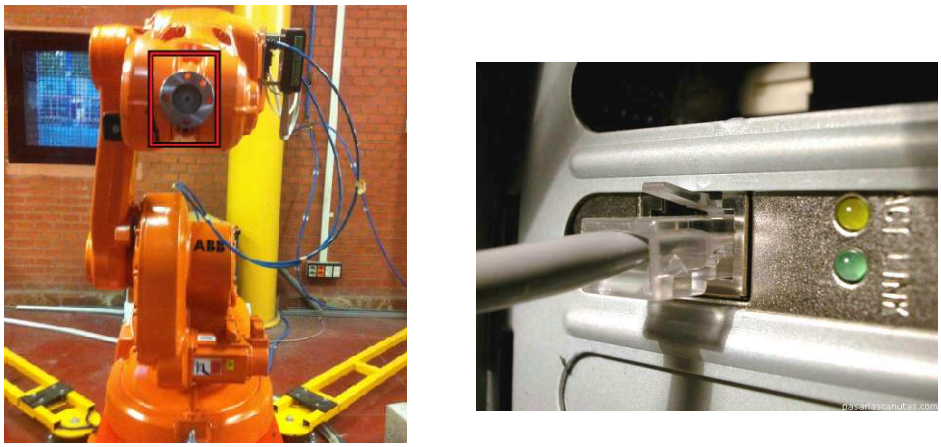


Fig. 6.1. Conexión del sensor con el robot y el ordenador.

- Una vez se asegura la conexión se procederá al encendido del robot girando el botón de la figura 6.2.

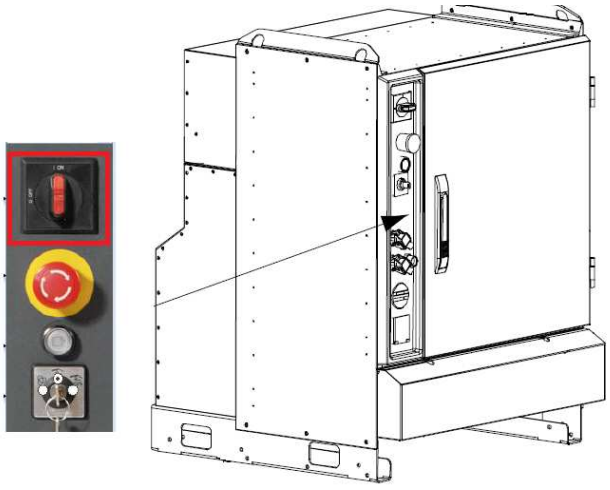


Fig. 6.2. Botón de encendido del robot.

- El robot ABB se puede colocar en modo automático o manual con el botón de la figura 6.3 dependiendo simplemente de si se desea mantener apretado el pulsador de seguridad ; para poder así permitir el movimiento del robot .

Fig. 6.3. Botón de selección de modo.

6.2. Puesta en marcha del software.

- Seleccionado el modo de funcionamiento; se ejecuta el programa de rapid con el botón de play de la figura 6.4 abriendo abierto el editor de programas con anterioridad.
  Con eso conseguimos que el robot se quede a la espera de recibir los datos del sensor.

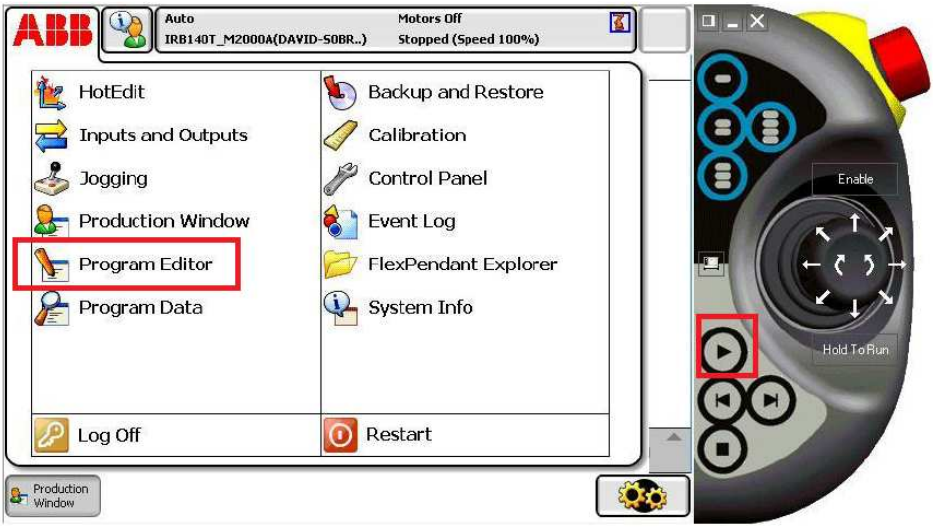


Fig. 6.4. Botón de inicio programa en Rapid.

- Después de esto se abre el ejecutable del programa de control del sensor JR3 y se pulsa el botón conectar como se ve en la figura 6.5 para así abrir el canal de comunicación entre el sensor y el ordenador.

Fig. 6.5. Botón de conectar.

Cuando se deseé empezar a recibir los datos se pulsa el botón de la figura 6.6 y estos se verán en la pantalla del ordenador y al mismo tiempo serán enviados al robot para que este inicie el movimiento. Los botones restantes se explican en el capítulo 4.4 de la interfaz del programa del JR3.

Fig. 6.6. Botón de play.

- Para acabar todas las conexiones primero se pulsa stop de la figura 6.7, parando la lectura de los datos del sensor y el botón de la figura 6.5 para desconectar y poder cerrar el programa.

Fig. 6.7. Botón de stop.

- Por último para finalizar la ejecución del programa en rapid se pulsa el botón que esta remarcado en la figura 6.8 y se apaga el robot girando el interruptor de la figura 5.2

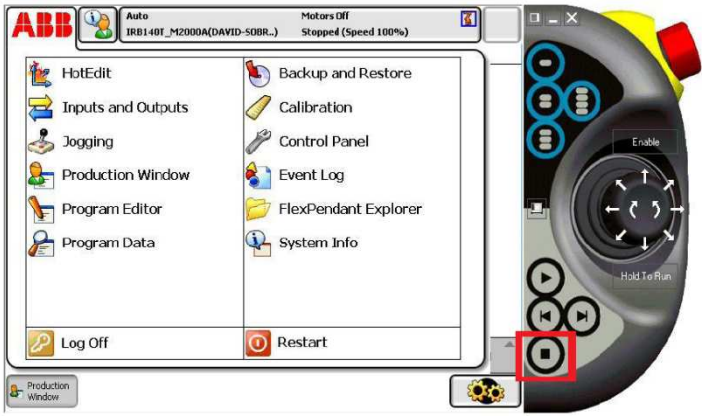


Fig. 6.8. Botón de finalizar programa en rapid.

7. EXPERIMENTOS Y COMPROBACIÓN DEL FUNCIONAMIENTO.

7.1. Medidas recogidas por el sensor JR3.

Para la comprobación del funcionamiento del sensor fuerza par JR3, se procede en este punto a ejercer sobre él fuerzas conocidas mediante pesas para estudiar así las medidas obtenidas. Se recogen en los siguientes puntos los valores obtenidos para pesas de 1, 3 y 5 kg en los distintos ejes de medida del sensor.

Los valores de escala aplicados en las lecturas son los configurados por defecto en la interfaz proporcionada por JR3.

Cabe destacar que para la realización de las medidas se ha realizado un reset justo antes de aplicar las fuerzas conocidas despreciando de este modo las masas ejercidas por el extremo del robot u otras partes que puedan afectar a las medidas.

7.1.1. Medida de fuerza en eje z.

Mediante la colocación de pesas situando el extremo del robot en posición vertical se pretenden observar los valores obtenidos en el eje z. En la figura 7.1 se muestra la colocación de la pesa para esta medición.

Fig. 7.1. Medida de fuerza en eje z.

- **Pesa de 1 kg.**

En primer lugar, se comienza ejerciendo sobre el sensor una fuerza conocida aproximada de 10N mediante una pesa de 1kg de la cual se obtienen los siguientes resultados representados en la figura 7.2.

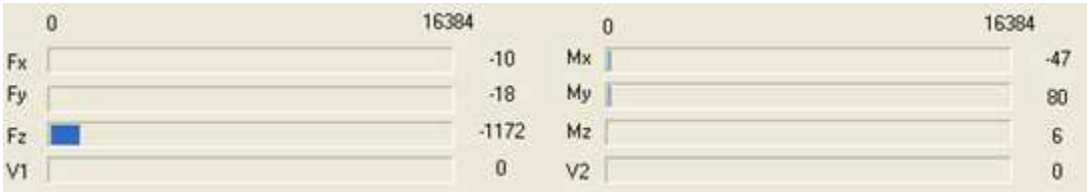


Fig. 7.2. Fuerza en eje z – 1 Kg.

Se observa que se obtiene un valor de -1172 debido a la escala que el programa de JR3 aplica por defecto. Debido a este primer valor obtenido se puede intuir que la escala aplicada al valor leído representa aproximadamente por cada Newton de fuerza, 100 unidades en la interfaz. El sentido lo indica el signo del valor obtenido ya que se trata de una fuerza en sentido negativo sobre el eje z del sensor JR3.

- **Pesa de 3 kg.**

Al igual que en el caso anterior, se procede ahora a aplicar sobre el extremo del robot una fuerza aproximada de 30N la cual proporciona los valores obtenido en la figura 7.3.

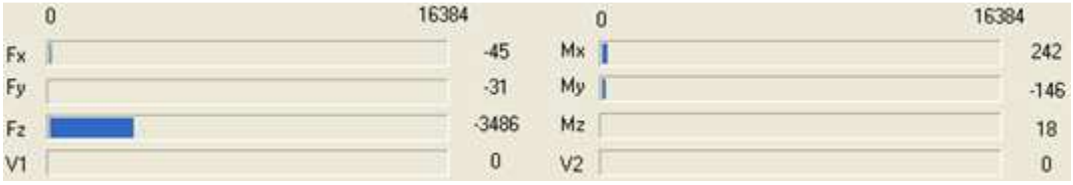


Fig. 7.3. Fuerza en eje z – 3 Kg.

El valor obtenido es de -3486 que tomando en cuenta la escala aplicada por defecto se aproxima al valor que se debería obtener.

- **Pesa de 6 kg.**

Como en los casos anteriores se representa en la figura 7.4 los valores obtenidos para una pesa de 6 kg.

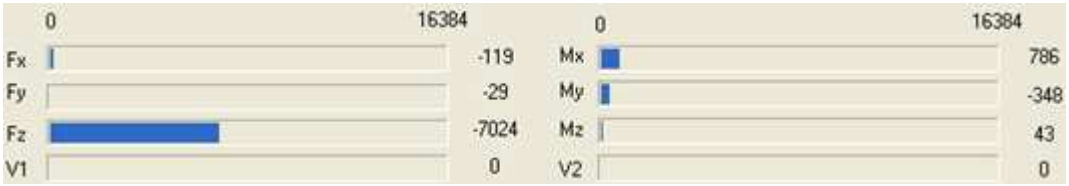


Fig. 7.4. Fuerza en eje z – 6 Kg.

Se observa que el valor obtenido es de -7024. Debido a que anteriormente se han ido obteniendo unos valores aproximados de 1100 por cada 10 N de fuerza aplicados, la suma de todos esos errores hacen que el valor se aleje un poco más del que se esperaba obtener.

En la tabla 7 1 se recogen todos los valores obtenidos para cada una de las tres pesas de 1, 3 y 6 kg. Debido a la sensibilidad del sensor se observa como a pesar de que la fuerza se aplica principalmente en el eje z, los demás ejes también soportan una fuerza aunque de menor magnitud.

PESO (Kg)	Fx	Fy	Fz	Mx	My	Mz
1	-10	-18	-1172	-47	80	6
3	-45	-31	-3486	242	-146	18
6	-119	-29	-7024	786	-348	43

Tabla 7 1. Valores de fuerza en eje z.

En la siguiente grafica (figura 7.5) se puede observar una variación casi lineal en las medidas del sensor mientras que se va aumentando el peso sobre el eje Z.

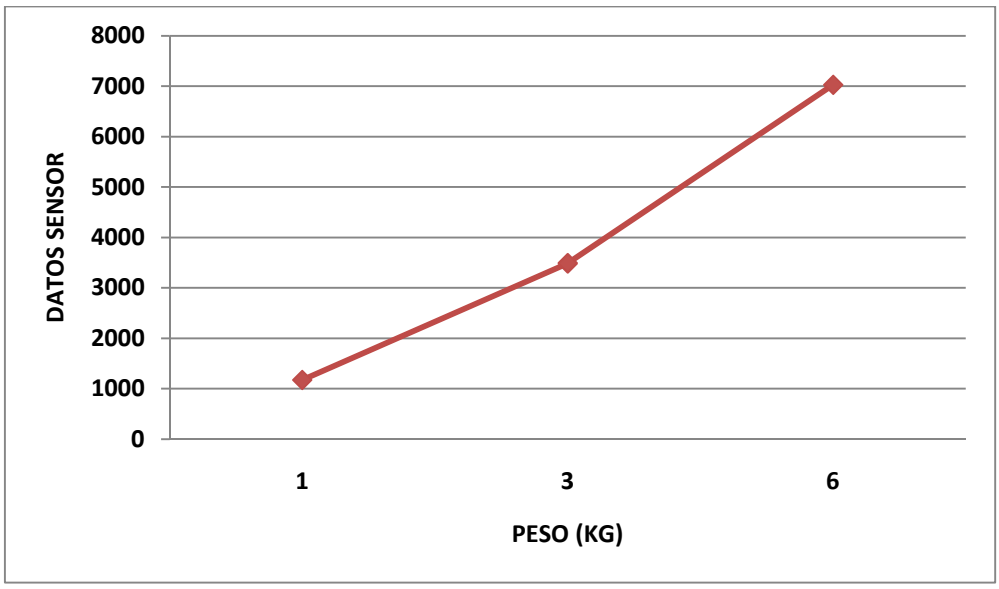


Fig. 7.5. Tabla datos sensor frente a peso eje Z.

7.1.2. Medida de fuerza en eje y – Par en eje x.

Para realizar las medidas de las fuerzas aplicadas en el eje y y el par en el eje x se sitúan las pesas de la manera que muestra la figura 7.6. Debido a la forma de sujeción del extremo del robot con el sensor mediante los tornillos, estas medidas pueden sufrir algunas variaciones con los valores realmente esperados.

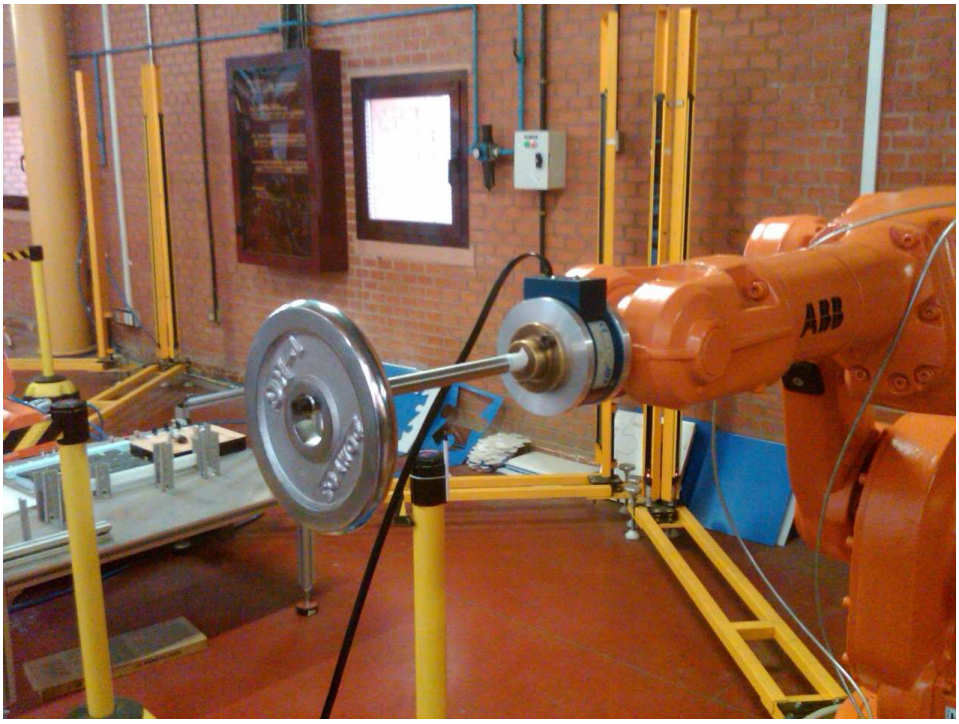


Fig. 7.6. Medida de fuerza en eje y – Par en eje x.

- **Pesa de 1 kg.**

En la figura 7.7 se ven los resultados obtenidos al situar en el extremo del robot una pesa de 1 kg que ejerce la mayor parte de la fuerza sobre el eje y. También, por el hecho de situarse en el extremo, ejerce un par sobre el extremo del robot en el eje x.

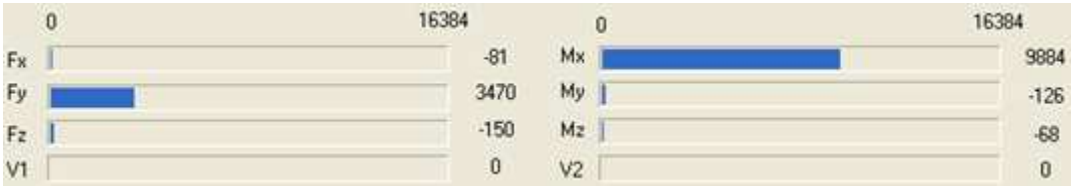


Fig. 7.7. Fuerza en eje y – Par en eje x – 1 Kg.

Se observa que el valor de fuerza obtenido es de 3470. Como se explica anteriormente, esto se debe a que debido a la forma de las chapas que sujetan el sensor y de la herramienta utilizada en el robot, la fuerza ejercida en el extremo afecta al sensor en otra magnitud. De igual manera, el efecto del par no proporciona valores que realmente demuestren que son datos reales.

- **Pesa de 3 kg.**

En la figura 7.8 se muestran los valores para una masa en este caso de 3 kg.

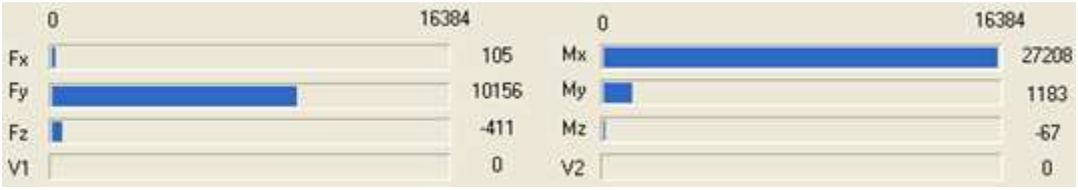


Fig. 7.8. Fuerza en eje y – Par en eje x – 3 Kg.

Destacar de esta imagen principalmente el hecho de que los valores obtenidos son aproximadamente el triple de los leídos con 1 kg, es decir, aumentan de marera proporcional correctamente.

- **Pesa de 6 kg.**

En la figura 7.9 se muestran los valores obtenidos para el caso de 6 kg.

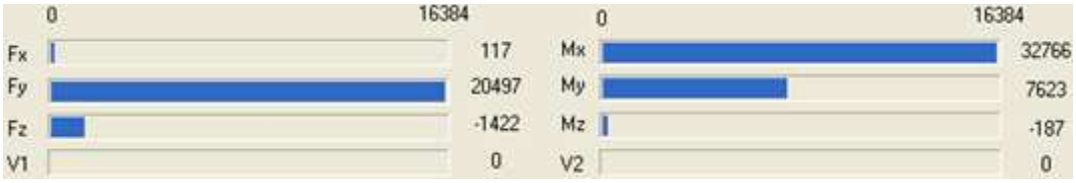


Fig. 7.9. Fuerza en eje y – Par en eje x – 6 Kg.

De nuevo en este caso se ve como la fuerza y el par aumentan de manera proporcional a la masa que se sitúa en el extremo.

En la tabla 7 2 se ven reflejado todos los valores obtenidos al aplicar la fuerza en el eje y y el par en el eje x.

PESO (Kg)	Fx	Fy	Fz	Mx	My	Mz
1	-81	3470	-150	9884	-126	-68
3	105	10156	-411	27208	1183	-67
6	117	20497	-1422	32766	7623	-187

Tabla 7 2. Valores de fuerza en eje y – Par en eje x.

Con la figura 6.10 se puede observar una variación casi lineal de los datos enviados por el sensor a medida que se va aumentando en peso sobre el eje Y.

Los valores aquí obtenidos reflejan un crecimiento prácticamente lineal, sobre todo en la fuerza, puesto que en los valores del momento intervienen otros factores produciéndose así pequeñas desviaciones en los datos (Véase figura 7.10).

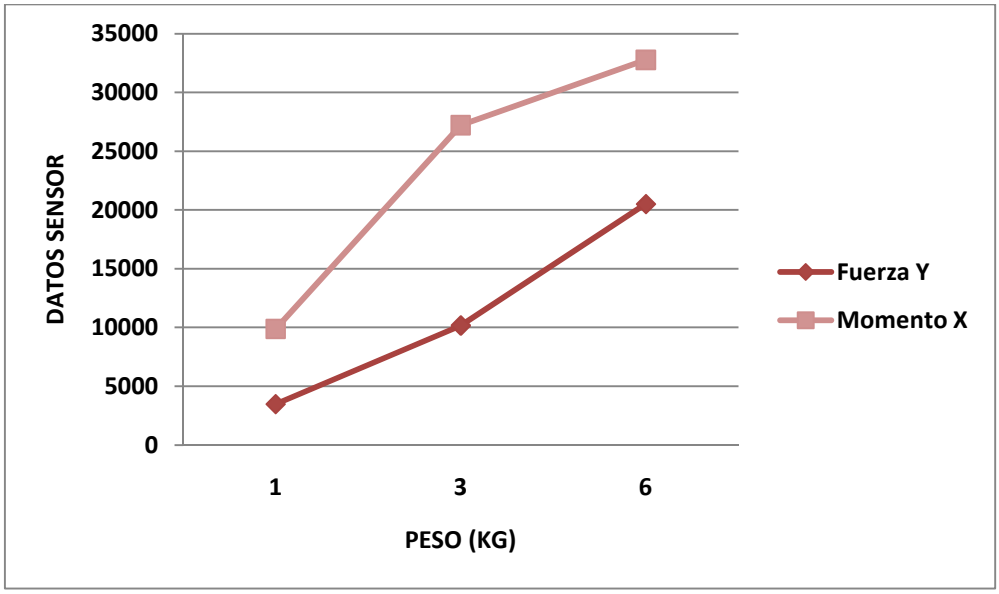


Fig. 7.10. Tabla fuerza Y, momento X frente a peso.

7.1.3. Medida de fuerza en eje x – Par en eje y.

Muy similar a los valores recogidos por el sensor en el punto anterior ya que la única diferencia de este caso con el que le precede es el eje sobre el que se aplica las fuerzas. En la figura 7.11 se muestra como se toman los datos en este caso.

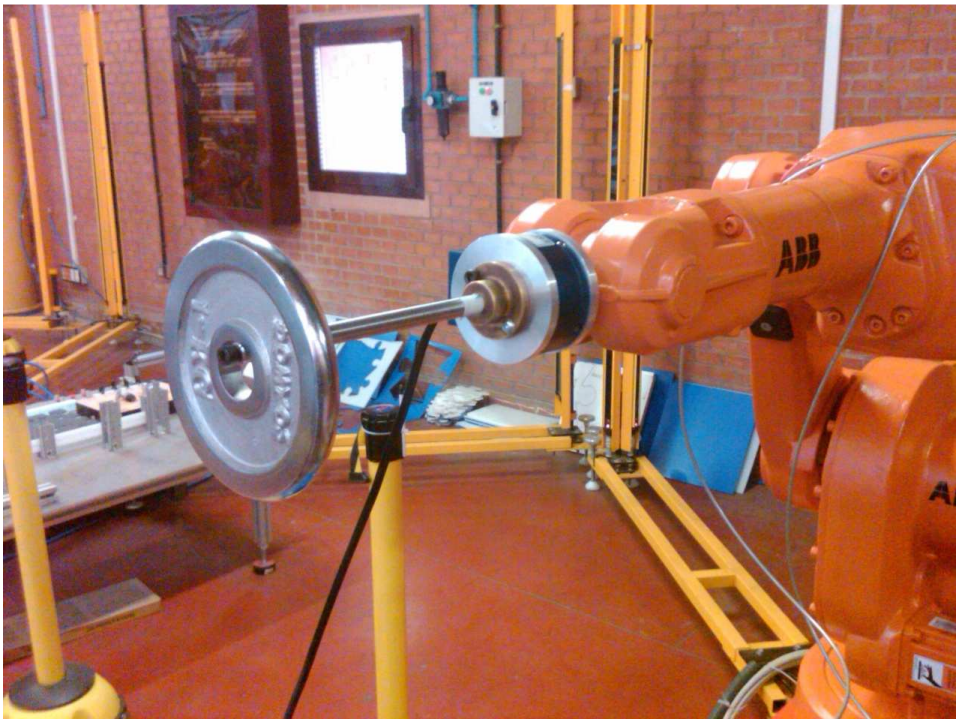


Fig. 7.11. Medida de fuerza en eje x – Par en eje y.

- **Pesa de 1 kg.**

Como en los casos anteriores, en la figura 7.12 se muestran los valores obtenidos para la masa de 1 Kg.

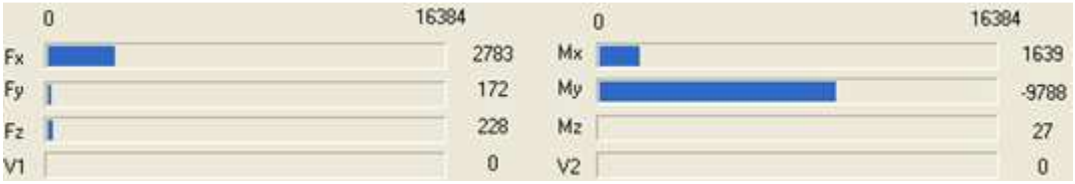


Fig. 7.12. Fuerza en eje x – Par en eje y – 1 Kg.

Como en el caso anterior, la fuerza y el par leídos son similares. Nuevamente se ve que debido a la forma de sujeción del sensor con la herramienta del robot la fuerza ejercida en el extremo no llega a ser la esperada obtenida en el sensor aunque si se asemeja.

- **Pesa de 3 kg.**

Se muestran en la figura 7.13 los valores obtenidos para una masa de 3 Kg situada en el extremo de la herramienta del robot.

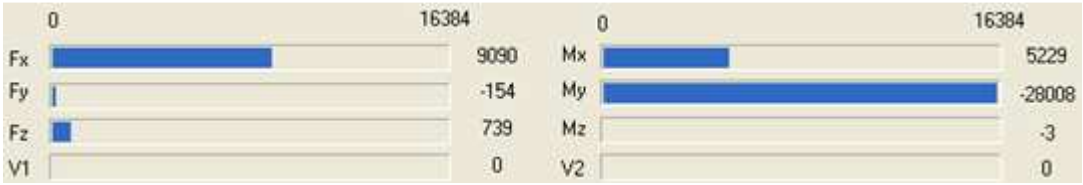


Fig. 7.13. Fuerza en eje x – Par en eje y – 3 Kg.

Los valores obtenidos crecen proporcionalmente. Cabe destacar también la influencia del valor de Mx que debido a la dificultad de situar el par únicamente sobre el eje y, comienza a tomar valores importantes que se deben tener en cuenta.

- **Pesa de 6 kg.**

Valores para masa de 6 kg representados en la figura 7.14.

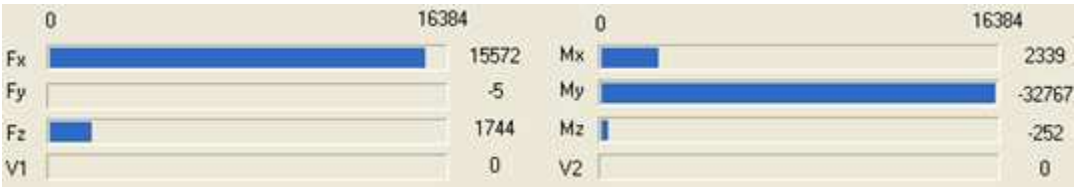


Fig. 7.14. Fuerza en eje x – Par en eje y – 6 Kg.

Valores normales obtenidos por el sensor de los cuales cabe destacar la pequeña corrección del par aplicado sobre el eje x para la realización de esta medida.

En la tabla 7 3 se muestra recogido un resumen de los valores obtenidos en este caso.

PESO (Kg)	Fx	Fy	Fz	Mx	My	Mz
1	2783	172	228	1639	-9788	27
3	9090	-154	739	5229	-28008	-3
6	15573	-5	1744	2339	-32767	-252

Tabla 7 3. Valores de fuerza en eje x – Par en eje y.

Con la figura 7.15 se puede observar una variación casi lineal en las medidas del sensor mientras se va aumentando el peso sobre el eje Z.

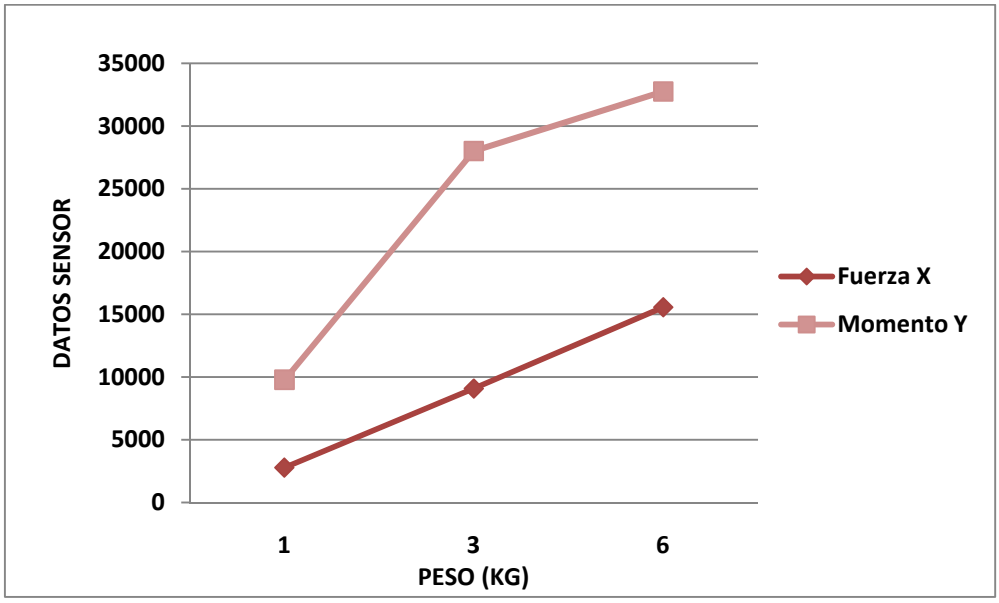


Fig. 7.15. Medidas de fuerza eje X, momento Y frente a peso

Los resultados son muy parecidos a la gráfica anterior figura 6.14 pudiendo sacar así las mismas conclusiones.

7.1.4. Par en eje z.

En las siguientes medidas se procede a situar en el extremo de la herramienta del robot ABB una varilla perpendicular a la herramienta la cual permita aplicar un par al extremo del sensor y leer los datos obtenidos. En la figura 7.16 se observa cómo se coloca dicha varilla para la captura de los datos.

Fig. 7.16. Medida del Par en eje z.

- **Pesa de 1 kg.**

En la figura 7.17 se observan los resultados obtenidos para masa de 1 kg en este caso.

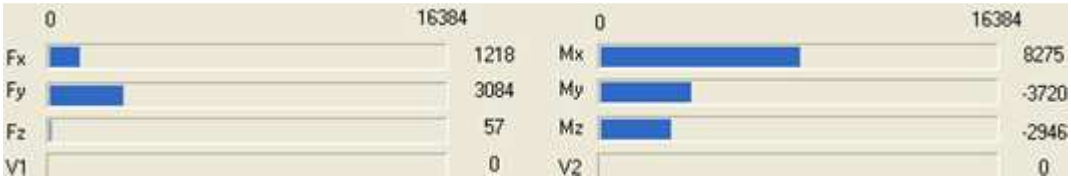


Fig. 7.17. Par en eje z – 1 Kg.

Se observa como los valores de Mx y My también toman valores importantes ya que al aplicar el par sobre el eje z, inevitablemente también se aplica un par en los otros dos ejes.

- **Pesa de 3 kg.**

Para una masa de 3 kg se representan en la figura 7.18 los valores obtenidos.

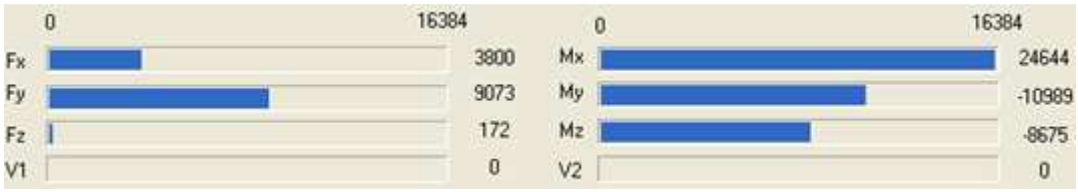


Fig. 7.18. Par en eje z – 3 Kg.

De nuevo cabe resaltar la proporcionalidad con la que los valores de par aumentan proporcionalmente a la masa.

- **Pesa de 6 kg.**

Valores obtenidos para una masa de 6 kg representados en la figura 7.19.

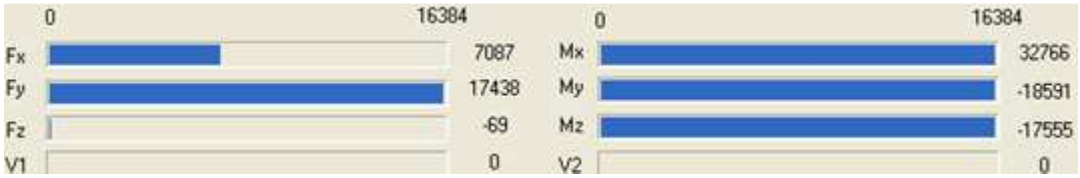


Fig. 7.19. Par en eje z – 6 Kg.

Se observan valores aparentemente normales teniendo en cuenta las situaciones descritas anteriormente.

En la tabla 3 4 se resumen todos los valores obtenido en este punto.

PESO (Kg)	Fx	Fy	Fz	Mx	My	Mz
1	1218	3084	57	8275	-3720	-2946
3	3800	9073	172	24644	-10989	-8675
6	7087	17438	-69	32766	-18591	-17555

Tabla 7 4. Valores de Par en eje z.

Con la figura 7.20 se puede observar una variación casi lineal en las medidas del sensor mientras se va aumentando el peso sobre el eje Z.

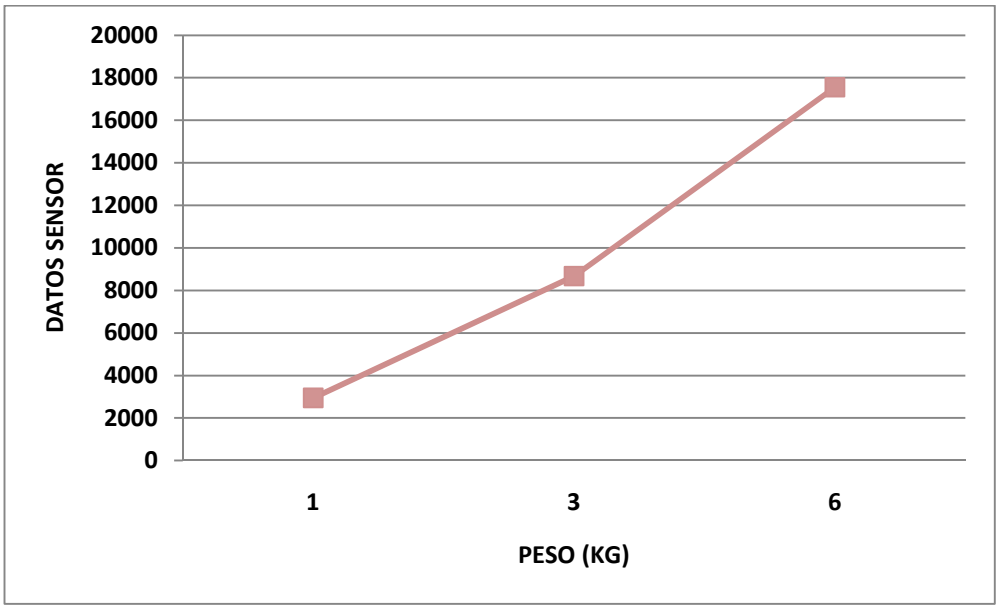


Fig. 7.20. Medidas de fuerza eje Z frente a peso.

7.1.5. Conclusiones

Después de haber realizado las diferentes pruebas en el sensor colocando los varios tipos de pesos en los ejes del sensor y analizando las distintas gráficas, se puede llegar a una clara conclusión en que a pesar de los pequeños errores de medición el sensor JR3 está perfectamente calibrado pudiendo tener absoluta confianza en la veracidad en los valores que son enviados por este.

7.2. Análisis de movimientos.

En este punto se estudiara todo el proceso de tratamiento de los datos desde que son enviados hasta que se ejecutan; pudiendo obtener el tiempo de respuesta y duración del movimiento del robot y la frecuencia de envió de datos.

La obtención de estos valores es de gran importancia para trabajos futuros o ampliaciones, puesto que se tendría una referencia de tiempos desde la que partir para optimizar el manejo del robot.

Se analizan los movimientos que ejecuta el robot en los tres ejes de coordenadas de la figura 7.21 y en las dos sentidos de forma independiente de los demás ejes y posteriormente se compararan obteniendo una visión global de los resultados.

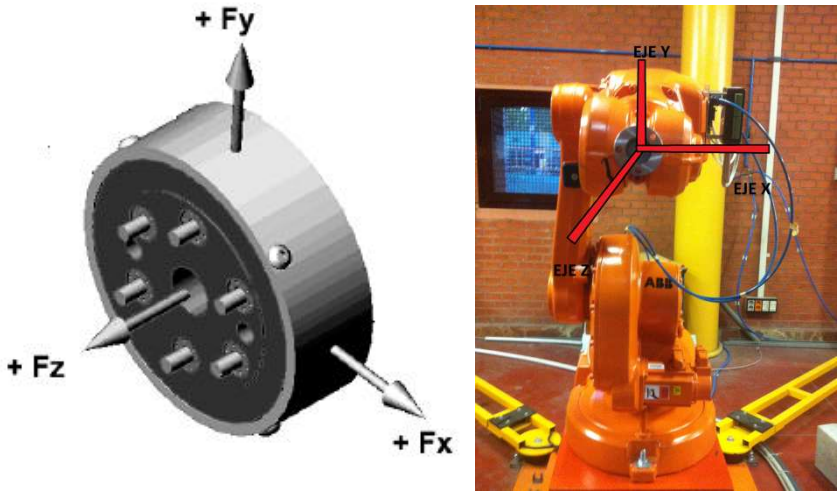


Fig. 7.21. Direcciones de desplazamiento del robot.

A continuación se procede al estudio de los movimientos, ayudados del archivo .txt que genera el código con los valores de fuerza leídos y la utilización de gráficas y tablas para una mayor comprensión.

7.2.2. Movimientos sobre el eje X.

Los movimientos estudiados son los que se producen a lo largo del eje X como muestra la figura 7.22.

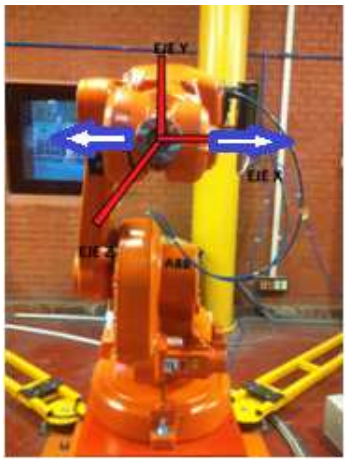


Fig. 7.22. Desplazamiento sobre eje X.

7.2.2.2 Desplazamiento en sentido positivo, eje x.

En la muestra de datos leídos por el sensor de la figura 7.23 se resaltan los valores de la fuerza aplicada en el eje x. El programa selecciona solo los valores superiores a 150 para ser enviados puesto que el resto son errores de lectura.

Para evitar las constantes salidas de rengo en los movimientos al enviarle datos muy grandes al robot, el valor máximo que se enviara será de 13000 para todos los datos leídos que sean superiores a dicho valor.

Se muestra la fuerza en los otros dos ejes , porque aunque teóricamente debería de ser 0 en la práctica resulta imposible realizar un movimiento perfecto sobre el eje x .

Fx			Fy			Fz
3	0	0	-3	0	0	1
-2	0	0	8	0	0	-4
19	0	0	15	0	0	3
6	0	0	-10	0	0	3
20	0	0	7	0	0	3
2	0	0	-5	0	0	1
1	0	0	5	0	0	5
0	0	0	-5	0	0	2
-1	0	0	-13	0	0	4
-4	0	0	-12	0	0	3
-1	0	0	-11	0	0	1
828	0	0	-250	0	0	-13
23510	0	0	-6631	0	0	1279
25656	0	0	-7688	0	0	1172
25469	0	0	-7184	0	0	1086
25459	0	0	-7204	0	0	1045
24968	0	0	-7042	0	0	1215
24734	0	0	-6871	0	0	1292
22763	0	0	-4109	0	0	1693
24064	0	0	178	0	0	1809
20775	0	0	3956	0	0	1698
20024	0	0	5073	0	0	1523
19017	0	0	9198	0	0	1700
17886	0	0	11000	0	0	1842
26516	0	0	-3998	0	0	845
26445	0	0	-13140	0	0	274
21614	0	0	-9909	0	0	720
193	0	0	-134	0	0	-173

Fig. 7.23. Valores desplazamientos sentido positivo eje X.

De forma teórica se puede calcular el tiempo que tarda en recorrer los valores enmarcados y la frecuencia de envió de estos.

Desde el programa en Rapid se puede conocer la posición inicial de la que parte el robot y la posición final al acabar el movimiento. Obteniendo así la distancia que recorre el robot en milímetros reflejados en la tabla 7.5.

	X	Y	Z
Punto inicial	1019.11	83.55	963.85
Punto final	471.52	24.35	918.95
Inicial - Final	547.59	59.20	44.90

Tabla 7 5. Distancia total en milímetros del desplazamiento.

Con una distancia calculada de 548 mm y sabiendo que la velocidad del robot puesta por software es de 100 mm/s obtenemos el tiempo total del desplazamiento tabla 7.6.

Distancia (mm)	Velocidad (m/s)	**TIEMPO (s)**
548	100	**5,48**

Tabla 7 6. Tiempo total del movimiento.

Sabiendo el tiempo del desplazamiento y dividiéndolo entre el total de datos, calculamos la frecuencia de envió, tabla 7.7.

Tiempo (s)	Numero de datos	**Frecuencia de envió (s)**
5,48	16	**0,3425**

Tabla 7 7. Frecuencia de envió de datos.

Una vez obtenidos los datos teóricos se estudiaran los prácticos para un posterior análisis del movimiento.

En la tabla 7.8 se puede ver un resumen de cuáles son los valores de fuerza en el eje x leído por el sensor, cuales son valores que el programa envía y la distancia en milímetros que se desplaza en robot en proporción a la fuerza aplicada.

El desplazamiento en milímetros es calculando por el programa dividiendo los datos recibidos entre 100.

En la última columna queda reflejado el tiempo que tarda el robot en ejecutar cada movimiento. Para calcularlo simplemente se ha dividido la distancia en milímetros ente la velocidad del robot que es de 100 mm/s.

Los valores que se encuentran enmarcados son aquellos que el robot tiene en cuenta; puesto que no se utiliza un valor de fuerza hasta que termine de ejecutarse el valor anterior.

Nº	**Leídos por el sensor**	**Enviados al robot**	**Distancia en (mm) del movimiento a realizar**	**Tiempo (s) que tarda en ejecutar el movimiento**
1	828,00	828,00	8,28	0,08
2	23510,00	13000,00	130,00	1,30
3	25656,00	13000,00	130,00	1,30
4	25469,00	13000,00	130,00	1,30
5	25459,00	13000,00	130,00	1,30
6	24968,00	13000,00	130,00	1,30
7	24734,00	13000,00	130,00	1,30
8	22763,00	13000,00	130,00	1,30
9	24064,00	13000,00	130,00	1,30
10	20775,00	13000,00	130,00	1,30
11	20024,00	13000,00	130,00	1,30
12	19017,00	13000,00	130,00	1,30
13	17886,00	13000,00	130,00	1,30
14	26516,00	13000,00	130,00	1,30
15	26445,00	13000,00	130,00	1,30
16	21614,00	13000,00	130,00	1,30

Tabla 7 8. Datos enviados, tiempo y distancia realizados por el robot eje X positivo.

A continuación se mostrará gráficamente todos los movimientos que ejecuta el robot a lo largo de su trayectoria.

Con la figura 7.24 se representa la longitud del movimiento que realiza el robot y el tiempo que tarda en ejecutar ese desplazamiento.

Con las barras verticales se representa cada cuanto tiempo el robot recibe un dato. Y dependiendo de la longitud de estas el dato es de mayor o menos valor.

Las barras horizontales son la duración de cada movimiento realizado; es decir; cuando se recibe un dato (barra vertical) el robot empieza a moverse, y se encuentra en esa situación hasta que termina de realizar ese movimiento. Y es en ese momento cuando procesa el siguiente dato enviado, depreciando los anteriores.

Cuando el movimiento es muy corto y termina de realizarse antes de que reciba siguiente valor el robot vuelve a ejecutar el movimiento que se realizó anteriormente.

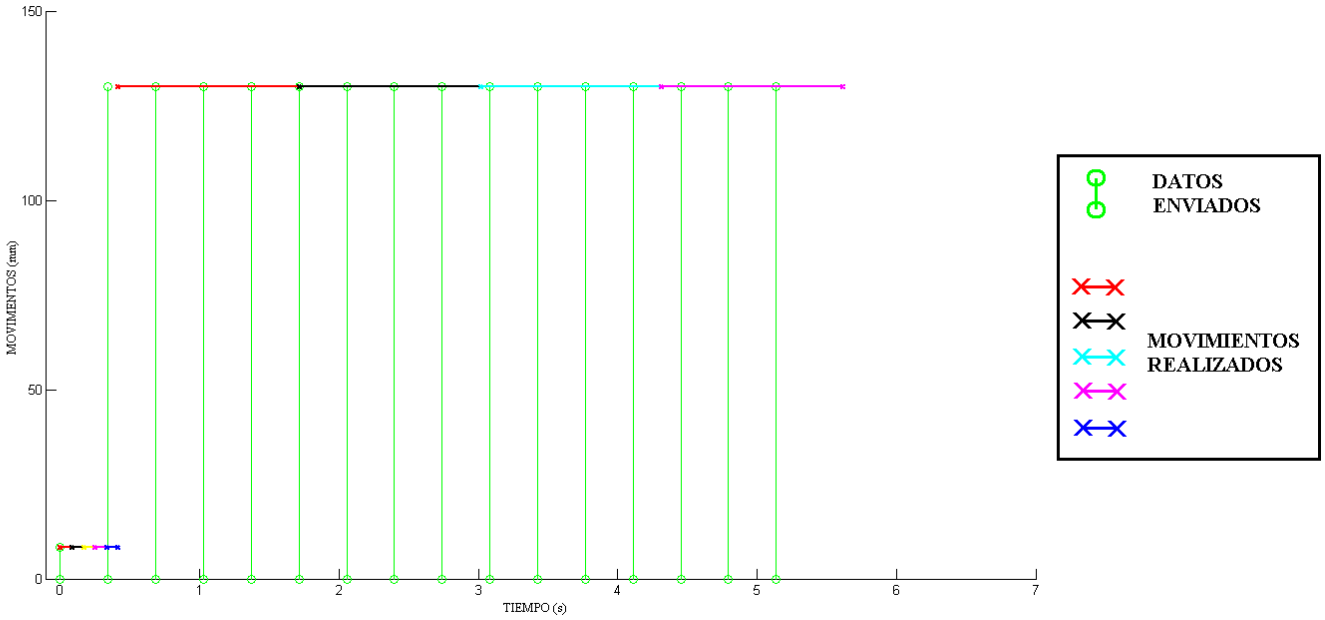


Fig. 7.24. Análisis de los valores de fuerza usados eje X positivo.

Una vez estudiado la figura 7.24 se comprueba un desplazamiento lineal y continuo menos en el valor inicial, que al ser demasiado pequeño en comparación con los demás , y por tanto realizarse de forma rápida ,le da tiempo a ejecutarse 5 veces hasta que se recibe el segundo dato.

Para concluir se puede decir que el desplazamiento real que realiza el robot, con respecto al de la suma de los datos enviados, que han sido ejecutados son muy parecidos , como se puede apreciar en la tabla 6.9. ; haciendo por tanto un desplazamiento preciso.

Suma de datos(mm)	Distancia recorrida (mm)
561.1	548

Tabla 7 9. Desplazamiento real y datos enviados

7.2.2.3 Desplazamiento en sentido negativo, eje x.

Los valores con los que se hacen el ensayo son los enmarcados en la 7.25.

Fx			Fy			Fz
-7			-20			-4
-11			-6			-1
-22			1			1
2 2			-9			-5
0 2			-9			2
3 2			-7			-3
8 2			-10			1
3 2			-13			-5
6 2			3 3			0
-13			-19			3
4 3			-4			-3
0 3			-7			-3
-3			-2			5
-16			6			2
-17622			2001			-288
-23088			1133			-51
-26405			1131			73
-27350			1291			97
-27548			1589			98
-27099			1694			58
-27294			1701			95
-28320			1552			48
-28546			1537			72
-24934			649			146
-26749			770			63
-29766			1096			-51
-29972			1404			101
-28220			2375			91

Fig. 7.25. Valores desplazamientos sentido negativo eje X.

Desde el programa de control del robot se puede conocer la posición inicial de la que parte el robot y la posición final al acabar el movimiento. Obteniendo así la distancia que recorre el robot en milímetros reflejados en la tabla 7.10.

	x	Y	Z
Punto inicial	460.96	14.65	898.23
Punto final	1019.11	-83.55	963.85
Inicial – Final	-558.15	98.20	-65.62

Tabla 7 10. Distancia total en milímetros del desplazamiento.

Con una distancia calculada de -558 mm y sabiendo que la velocidad del robot puesta por software es de 100 mm/s obtenemos el tiempo total del desplazamiento figura 7.11.

Distancia (mm)	Velocidad (m/s)	**TIEMPO (s)**
558	100	**5,58**

Tabla 7 11. Tiempo total del movimiento.

Sabiendo el tiempo del desplazamiento y dividiéndolo entre el total de datos, calculamos la frecuencia de envió, tabla 7.12.

Tiempo (s)	Numero de datos	**Frecuencia de envió (s)**
5,48	14	0,3985

Tabla 7 12. Frecuencia de envió de datos.

En la tabla 7.13 se muestra todos los datos enviados al robot una vez procesados, así como la distancia que debe de desplazarse y el tiempo que tarda en realizar dicho desplazamiento.

Nº	**Leídos por el sensor**	**Enviados al robot**	**Distancia en (mm) del movimiento a realizar**	**Tiempo (s) que tarda en ejecutar el movimiento**
1	-17622,00	-13000,00	-130,00	1,30
2	-23088,00	-13000,00	-130,00	1,30
3	-26405,00	-13000,00	-130,00	1,30
4	-27350,00	-13000,00	-130,00	1,30
5	-27548,00	-13000,00	-130,00	1,30
6	-27099,00	-13000,00	-130,00	1,30
7	-27294,00	-13000,00	-130,00	1,30
8	-28320,00	-13000,00	-130,00	1,30
9	-28546,00	-13000,00	-130,00	1,30
10	-24934,00	-13000,00	-130,00	1,30
11	-26749,00	-13000,00	-130,00	1,30
12	-29766,00	-13000,00	-130,00	1,30
13	-29972,00	-13000,00	-130,00	1,30
14	-28220,00	-13000,00	-130,00	1,30

Tabla 7 13. Datos enviados, tiempo y distancia realizados por el robot eje X negativo.

En la figura 7.26 vemos una ejecución del movimiento total mente lineal, a diferencia del caso positivo puesto que este presentaba un valor diferente al resto.

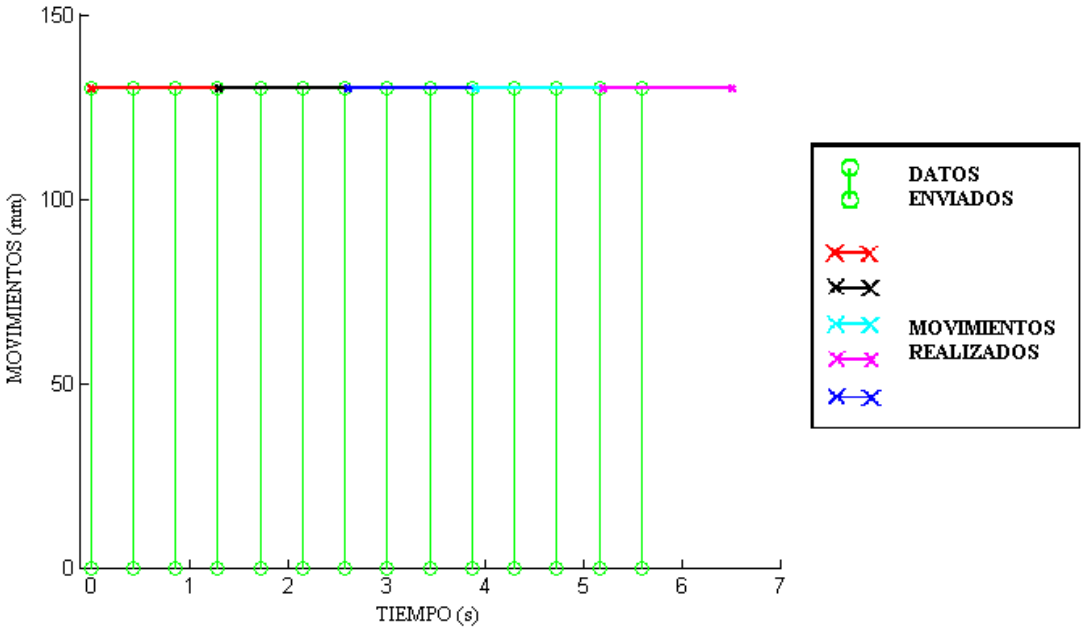


Fig. 7.26. Análisis de los valores de fuerza usados, eje X negativo.

Se puede comprobar que solo se ejecuten cuatro valores de los 14 que se envían puesto, que no da tiempo a ejecutarlos todos.

El desplazamiento real que realiza el robot, con respecto al de la suma de los datos enviados, y ejecutados son muy parecidos ; como se puede apreciar en la tabla 6.14 haciendo por tanto un desplazamiento preciso.

Suma de datos(mm)	Distancia recorrida (mm)
520	558

Tabla 7 14. Desplazamiento real y datos enviados

7.2.3. Movimiento sobre el eje Y.

Los movimientos estudiados son los que se producen a lo largo del eje Y como muestra la figura 7.27.

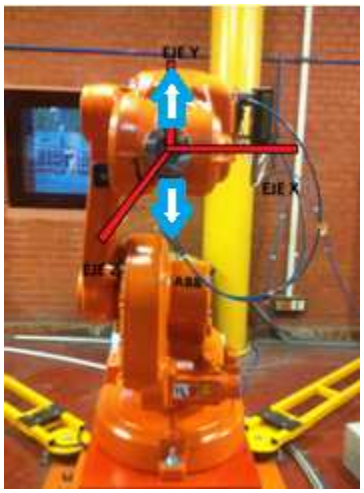


Fig. 7.27. Movimiento sobre el eje Y.

7.2.3.2 Desplazamiento en sentido positivo, eje y.

Los valores con los que se hacen el ensayo son los enmarcados en la 7.28.

-658	0	0	9907	0	0	79
-1924	0	0	26722	0	0	-349
-1889	0	0	27596	0	0	-314
-2077	0	0	29375	0	0	-416
-1798	0	0	29646	0	0	-400
-1497	0	0	28357	0	0	-340
-1896	0	0	28873	0	0	-377
-1583	0	0	28202	0	0	-229
-2078	0	0	31754	0	0	-454
-1945	0	0	30309	0	0	-179
502	0	0	21811	0	0	703
4195	0	0	18721	0	0	826
3505	0	0	24635	0	0	7
4009	0	0	22904	0	0	102
5624	0	0	23443	0	0	-289
-1335	0	0	24247	0	0	-337
-6283	0	0	26409	0	0	-547
-1159	0	0	26456	0	0	84
-7774	0	0	30719	0	0	-748
-8025	0	0	28988	0	0	-333
189	0	0	30566	0	0	-155
6412	0	0	27493	0	0	-537
11034	0	0	24554	0	0	-525
14146	0	0	21504	0	0	-574
16080	0	0	22445	0	0	-1120
7033	0	0	18241	0	0	7
-524	0	0	21143	0	0	372
-1946	0	0	22611	0	0	482
-441	0	0	18122	0	0	13
-3911	0	0	23126	0	0	-296
-6648	0	0	29773	0	0	-675
-6930	0	0	24944	0	0	-237
-4055	0	0	12665	0	0	283

Fig. 7.28. Valores desplazamientos sentido positivo eje Y.

Desde el programa en Rapid se puede conocer la posición inicial de la que parte el robot y la posición final al acabar el movimiento. Obteniendo así la distancia que recorre el robot en milímetros reflejados en la tabla 7.15.

	x	y	z
Punto inicial	878.79	525.86	856.30
Punto final	849.62	-651.40	868.75
Inicial – Final	28.67	1177.26	-12.45

Tabla 7 15. Distancia total en milímetros del desplazamiento.

Con una distancia calculada de 1177 mm y sabiendo que la velocidad del robot puesta por software es de 100 mm/s obtenemos el tiempo total del desplazamiento, figura 7.16.

Distancia (mm)	Velocidad (m/s)	**TIEMPO (s)**
1177	100	**11,77**

Tabla 7 16. Tiempo total del movimiento.

Sabiendo el tiempo del desplazamiento y dividiéndolo entre el total de datos, calculamos la frecuencia de envió, tabla 7.17.

Tiempo (s)	Numero de datos	**Frecuencia de envió (s)**
11,77	33	0,3566

Tabla 7 17. Frecuencia de envió de datos

En la tabla 7.18 se muestra todos los datos enviados al robot una vez procesados, así como la distancia que debe de desplazarse y el tiempo que tarda en realizar dicho desplazamiento.

Dejando enmarcados aquellos que son utilizados por el robot.

Nº	Leídos por el sensor	Enviados al robot	Distancia en (mm) del movimiento a realizar	Tiempo (s) que tarda en ejecutar el movimiento
1	9907,00	9907,00	99,07	0,99
2	26722,00	13000,00	130,00	1,30
3	27596,00	13000,00	130,00	1,30
4	29375,00	13000,00	130,00	1,30
5	29646,00	13000,00	130,00	1,30
6	28357,00	13000,00	130,00	1,30
7	28873,00	13000,00	130,00	1,30
8	28202,00	13000,00	130,00	1,30
9	31754,00	13000,00	130,00	1,30
10	30309,00	13000,00	130,00	1,30
11	21811,00	13000,00	130,00	1,30
12	18721,00	13000,00	130,00	1,30
13	24635,00	13000,00	130,00	1,30
14	22904,00	13000,00	130,00	1,30
15	23443,00	13000,00	130,00	1,30
16	24247,00	13000,00	130,00	1,30
17	26409,00	13000,00	130,00	1,30
18	26456,00	13000,00	130,00	1,30
19	30719,00	13000,00	130,00	1,30
20	28988,00	13000,00	130,00	1,30
21	30566,00	13000,00	130,00	1,30
22	27493,00	13000,00	130,00	1,30
23	24554,00	13000,00	130,00	1,30
24	21504,00	13000,00	130,00	1,30
25	22445,00	13000,00	130,00	1,30
26	18241,00	13000,00	130,00	1,30
27	21143,00	13000,00	130,00	1,30
28	22611,00	13000,00	130,00	1,30
29	18122,00	13000,00	130,00	1,30
30	23126,00	13000,00	130,00	1,30
31	29773,00	13000,00	130,00	1,30
32	24944,00	13000,00	130,00	1,30
33	12665,00	12665,00	126,65	1,27

Tabla 7 18. Datos enviados, tiempo y distancia realizados por el robot eje X negativo.

En la figura 7.29 vemos una ejecución del movimiento prácticamente lineal, con dos valores de diferente modulo, coincidiendo con el principio y final de aplicación de la fuerza.

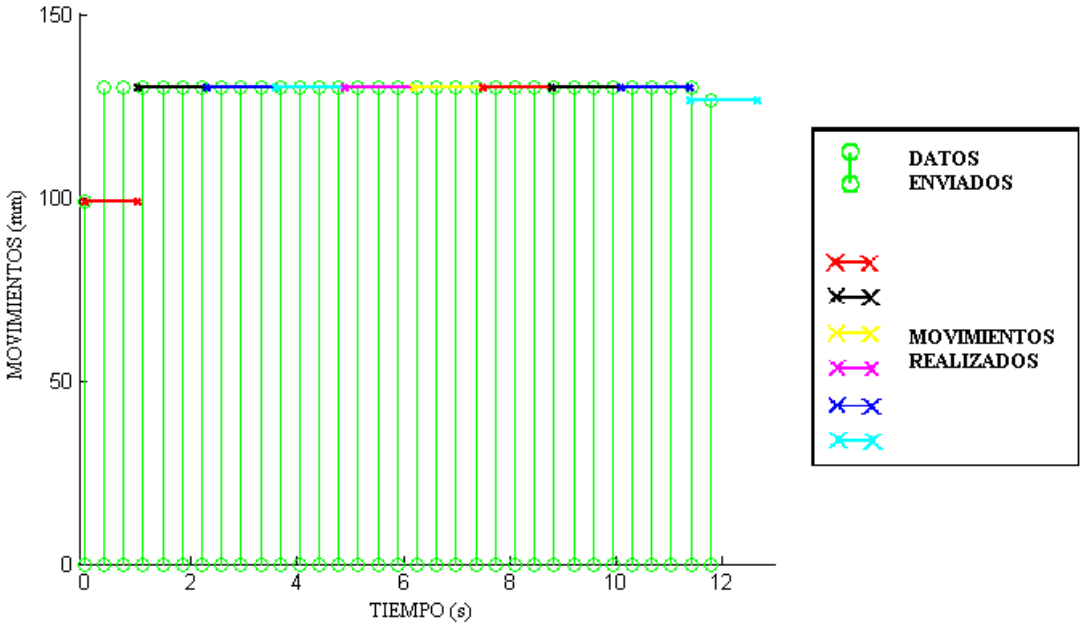


Fig. 7.29. Análisis de los valores de fuerza usados, eje Y positivo.

Se puede comprobar que se ejecuten 10 valores de los 33 que se envían, puesto que no da tiempo a ejecutarlos todos.

El desplazamiento real que realiza el robot, con respecto al de la suma de los datos enviados y ejecutados; son muy parecidos, como se puede apreciar en la tabla 7 19 ; haciendo por tanto un desplazamiento preciso.

Suma de datos(mm)	Distancia recorrida (mm)
1177	1270

Tabla 7 19. Desplazamiento real y datos enviados

7.2.4. Desplazamiento en sentido negativo , eje y.

Los valores con los que se hacen el ensayo son los enmarcados en la figura 7.30.

-32	0	0	-174	0	0	-34
59	0	0	-25430	0	0	-1781
-298	0	0	-29368	0	0	-2112
-469	0	0	-29372	0	0	-1999
82	0	0	-24378	0	0	-1143
-4088	0	0	-28565	0	0	-1111
-5442	0	0	-27780	0	0	-819
-5400	0	0	-27335	0	0	-648
-2505	0	0	-25415	0	0	-840
1062	0	0	-27875	0	0	-1455
4474	0	0	-30442	0	0	-2021
4391	0	0	-28799	0	0	-2156
2793	0	0	-26375	0	0	-1826
-2839	0	0	-26384	0	0	-1337
-5784	0	0	-26359	0	0	-1121
-6985	0	0	-26374	0	0	-1149
-5798	0	0	-24137	0	0	-966
-6201	0	0	-22510	0	0	-812
498	0	0	-22201	0	0	-1070
5139	0	0	-20296	0	0	-1087
8370	0	0	-20872	0	0	-1577
9708	0	0	-21934	0	0	-1603
9736	0	0	-21593	0	0	-1220
-2213	0	0	-21103	0	0	-1071
-3792	0	0	-17799	0	0	-702
-7525	0	0	-18328	0	0	-526
-5574	0	0	-7089	0	0	-70

Fig. 7.30. Valores desplazamientos sentido negativo eje Y.

Desde el programa en Rapid se puede conocer la posición inicial de la que parte el robot y la posición final al acabar el movimiento. Obteniendo así la distancia que recorre el robot en milímetros reflejados en la tabla 7 20.

	x	y	z
Punto inicial	806.76	708.31	718.79
Punto final	878.79	525.86	856.30
Inicial – Final	-72.03	182.45	-137.51

Tabla 7 20. Distancia total en milímetros del desplazamiento.

Con una distancia calculada de 182 mm y sabiendo que la velocidad del robot puesta por software es de 100 mm/s obtenemos el tiempo total del desplazamiento, tabla 7 21.

Distancia (mm)	Velocidad (m/s)	**TIEMPO (s)**
182	100	**1,82**

Tabla 7 21. Tiempo total del movimiento.

Sabiendo el tiempo del desplazamiento y dividiéndolo entre el total de datos, calculamos la frecuencia de envió, tabla 7 22.

Tiempo (s)	Numero de datos	**Frecuencia de envió (s)**
1,82	26	0,07

Tabla 7 22. Frecuencia de envió de datos

En la tabla 7 23 se muestra todos los datos enviados al robot una vez procesados, así como la distancia que debe de desplazarse y el tiempo que tarda en realizar dicho desplazamiento.

Dejando enmarcados aquellos que son utilizados por el robot.

Nº	**Leídos por el sensor**	**Enviados al robot**	**Distancia en (mm) del movimiento a realizar**	**Tiempo (s) que tarda en ejecutar el movimiento**
1	-25430,00	-13000,00	-130,00	1,30
2	-29368,00	-13000,00	-130,00	1,30
3	-29372,00	-13000,00	-130,00	1,30
4	-24378,00	-13000,00	-130,00	1,30
5	-28565,00	-13000,00	-130,00	1,30
6	-27780,00	-13000,00	-130,00	1,30
7	-27335,00	-13000,00	-130,00	1,30
8	-25415,00	-13000,00	-130,00	1,30
9	-27875,00	-13000,00	-130,00	1,30
10	-30442,00	-13000,00	-130,00	1,30
11	-28799,00	-13000,00	-130,00	1,30
12	-26375,00	-13000,00	-130,00	1,30
13	-26384,00	-13000,00	-130,00	1,30
14	-26359,00	-13000,00	-130,00	1,30
15	-26374,00	-13000,00	-130,00	1,30
16	-24137,00	-13000,00	-130,00	1,30
17	-22510,00	-13000,00	-130,00	1,30
18	-22201,00	-13000,00	-130,00	1,30
19	-20296,00	-13000,00	-130,00	1,30
20	-20872,00	-13000,00	-130,00	1,30
21	-21934,00	-13000,00	-130,00	1,30
22	-21593,00	-13000,00	-130,00	1,30
23	-21103,00	-13000,00	-130,00	1,30
24	-17799,00	-13000,00	-130,00	1,30
25	-18328,00	-13000,00	-130,00	1,30
26	-7089,00	-7089,00	-70,89	0,71

Tabla 7 23. Datos enviados, tiempo y distancia realizados por el robot eje Y negativo.

En la figura 7.31 se representan en un espacio muy corto de tiempo muchos valores de módulo máximo que requieren de mucho tiempo para ejecutarse; haciendo que de los 26 valores solo se usen 2.

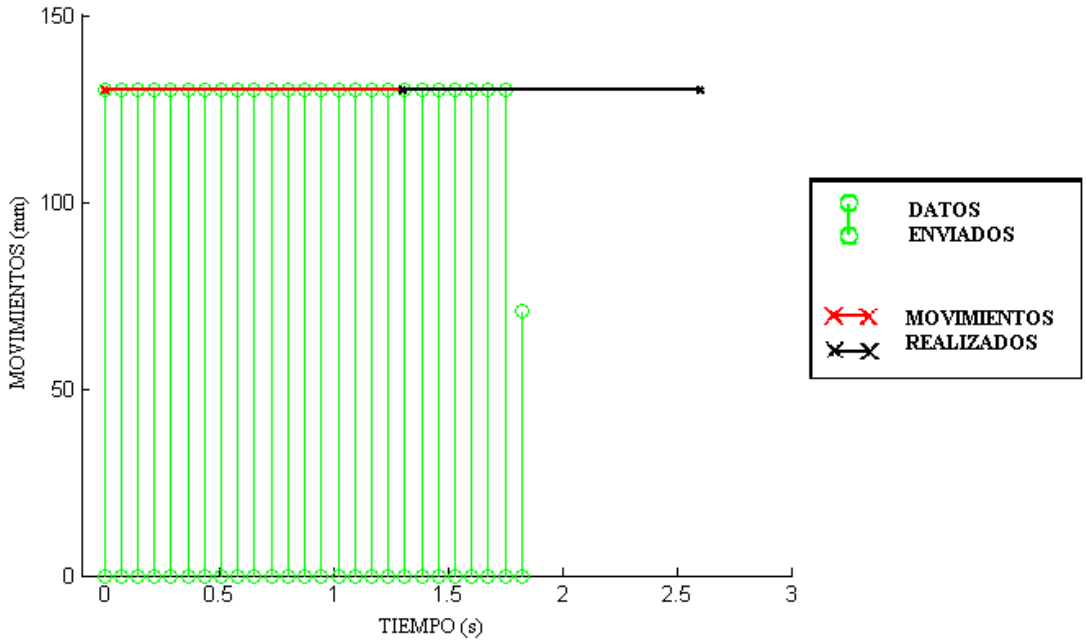


Fig. 7.31. Análisis de los valores de fuerza usados, eje Y negativo.

Este análisis es especial y no se tendrá en cuenta en las conclusiones finales debido su gran error de medición a la hora de calcular la posición final puesto que el brazo no realiza un desplazamiento solo en el eje Y; además al acabar el movimiento retrocede sobre su trayectoria.

7.2.5. Movimiento sobre el eje Z.

Los movimientos estudiados son los que se producen a lo largo del eje Z como muestra la figura 7.32.

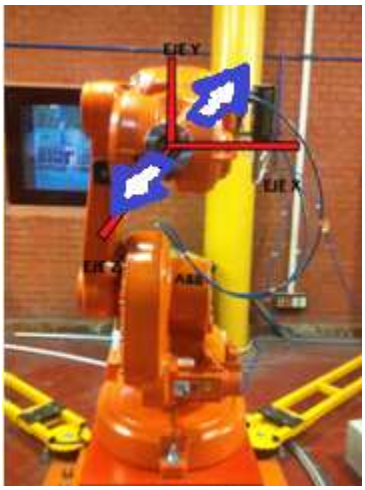


Fig. 7.32. Movimiento sobre el eje z.

7.2.6. Desplazamiento en sentido positivo, eje z.

Los valores con los que se hacen el ensayo son los enmarcados en la figura 7.33.

1626	0	0	116	0	0	9841
2433	0	0	-1248	0	0	11400
830	0	0	-46	0	0	11928
617	0	0	907	0	0	11634
1602	0	0	868	0	0	12146
2486	0	0	1026	0	0	12084
1591	0	0	1424	0	0	12522
1587	0	0	1655	0	0	12049
1481	0	0	2095	0	0	12435
1080	0	0	2452	0	0	12675
1185	0	0	2662	0	0	12632
1199	0	0	788	0	0	11408
1406	0	0	1410	0	0	12244
2617	0	0	1028	0	0	12434
2393	0	0	424	0	0	11879
1130	0	0	1251	0	0	11049
965	0	0	743	0	0	10658
384	0	0	787	0	0	10735
719	0	0	1029	0	0	9708
-133	0	0	975	0	0	8441
91	0	0	910	0	0	7772

Fig. 7.33. Valores desplazamientos sentido positivo eje Z.

Desde el programa en Rapid se puede conocer la posición inicial de la que parte el robot y la posición final al acabar el movimiento. Obteniendo así la distancia que recorre el robot en milímetros reflejados en la tabla 7 24.

	x	y	Z
Punto inicial	913.20	138.39	883.54
Punto final	848.86	-232.26	143.54
Inicial – Final	64.34	370.65	740

Tabla 7 24. Distancia total en milímetros del desplazamiento.

Con una distancia calculada de 740 mm y sabiendo que la velocidad del robot puesta por software es de 100 mm/s obtenemos el tiempo total del desplazamiento, figura 7 25.

Distancia (mm)	Velocidad (m/s)	**TIEMPO (s)**
740	100	**7,4**

Tabla 7 25. Tiempo total del movimiento.

Sabiendo el tiempo del desplazamiento y dividiéndolo entre el total de datos, calculamos la frecuencia de envió, tabla 7 26.

Tiempo (s)	Numero de datos	**Frecuencia de envió (s)**
7,4	21	**0,352**

Tabla 7 26. Frecuencia de envió de datos

En la tabla 7 27 se muestra todos los datos enviados al robot una vez procesados, así como la distancia que debe de desplazarse y el tiempo que tarda en realizar dicho desplazamiento.

Dejando enmarcados aquellos que son utilizados por el robot.

Nº	Leídos por el sensor	Enviados al robot	Distancia en (mm) del movimiento a realizar	Tiempo (s) que tarda en ejecutar el movimiento
1	9841,00	9841,00	98,41	0,98
2	11400,00	11400,00	114,00	1,14
3	11928,00	11928,00	119,28	1,19
4	11634,00	11634,00	116,34	1,16
5	12146,00	12146,00	121,46	1,21
6	12084,00	12084,00	120,84	1,21
7	12522,00	12522,00	125,22	1,25
8	12049,00	12049,00	120,49	1,20
9	12435,00	12435,00	124,35	1,24
10	12675,00	12675,00	126,75	1,27
11	12632,00	12632,00	126,32	1,26
12	11408,00	11408,00	114,08	1,14
13	12244,00	12244,00	122,44	1,22
14	12434,00	12434,00	124,34	1,24
15	11879,00	11879,00	118,79	1,19
16	11049,00	11049,00	110,49	1,10
17	10658,00	10658,00	106,58	1,07
18	10735,00	10735,00	107,35	1,07
19	9708,00	9708,00	97,08	0,97
20	8441,00	8441,00	84,41	0,84
21	7772,00	7772,00	77,72	0,78

Tabla 7 27. Datos enviados, tiempo y distancia realizados por el robot eje Z positivo.

En la figura 7.34 vemos una ejecución de un movimiento sobre un eje, el z, en el que es difícil ejercer un movimiento lineal como se ve en dicha figura, al tener un mal acceso a este produciendo valores de fuerza diferentes entre sí .

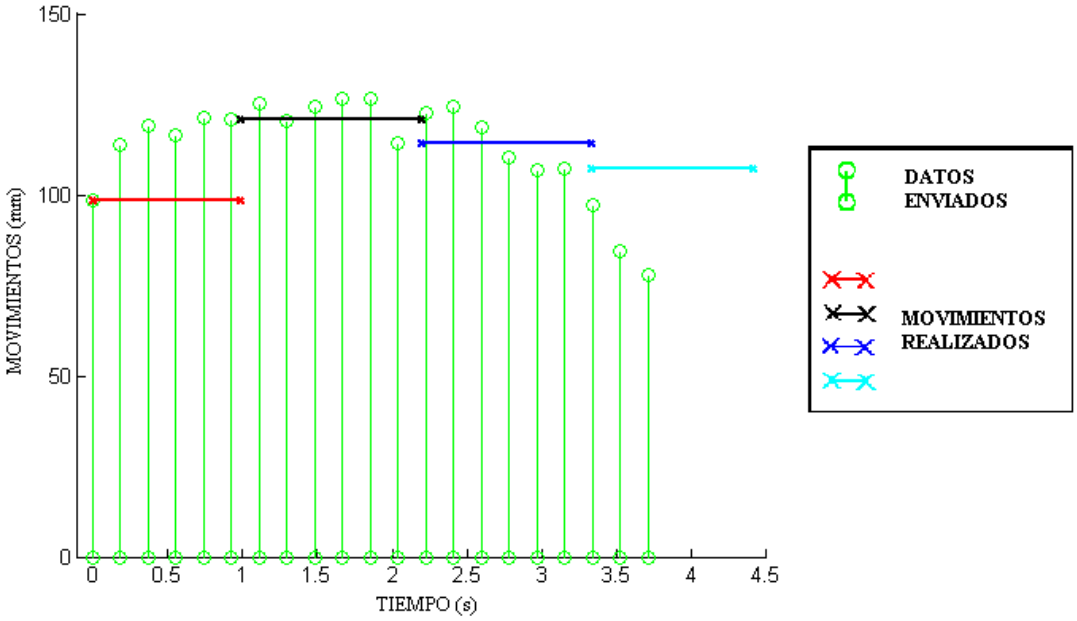


Fig. 7.34. Análisis de los valores de fuerza usados, eje Y negativo.

Se puede comprobar que se ejecuten 4 valores de los 21 que se envían, puesto que no da tiempo a ejecutarlos todos.

El desplazamiento real que realiza el robot, con respecto al de la suma de los datos enviados y ejecutados; es diferente, como se puede apreciar en la tabla 7 28 haciendo por tanto un desplazamiento poco preciso aunque posteriormente se explicara el porqué.

Suma de datos(mm)	Distancia recorrida (mm)
441	740

Tabla 7 28. Desplazamiento real y datos enviados

7.2.6.2 Desplazamiento en sentido negativo, eje z.

Los valores con los que se hacen el ensayo son los enmarcados en la 7.35.

1234	0	0	-620	0	0	-9867
3025	0	0	-1739	0	0	-14292
5742	0	0	-1899	0	0	-15964
5837	0	0	-2413	0	0	-15634
6128	0	0	-2648	0	0	-16257
6977	0	0	-2775	0	0	-17436
5848	0	0	-2038	0	0	-17431
6536	0	0	-2422	0	0	-17223
7024	0	0	-2363	0	0	-18116
6769	0	0	-2589	0	0	-17468
-703	0	0	581	0	0	-13487
-6155	0	0	1190	0	0	-14258
-11125	0	0	1869	0	0	-12944
-21702	0	0	1826	0	0	-13861
-18285	0	0	-1770	0	0	-12284
-7585	0	0	-4054	0	0	-12611
-5678	0	0	-1902	0	0	-12505
219	0	0	774	0	0	-11355
1773	0	0	642	0	0	-10135
3575	0	0	-885	0	0	-8781
2716	0	0	631	0	0	-9123
7251	0	0	2301	0	0	-9534
3165	0	0	2291	0	0	-8229
-3741	0	0	-2285	0	0	-7652
-923	0	0	874	0	0	-6948
-2931	0	0	-1800	0	0	-6173
441	0	0	-92	0	0	-8082
2907	0	0	-162	0	0	-9095
8128	0	0	-4088	0	0	-7599

Fig. 7.35. Valores desplazamientos sentido negativo eje Z

Desde el programa en Rapid se puede conocer la posición inicial de la que parte el robot y la posición final al acabar el movimiento. Obteniendo así la distancia que recorre el robot en milímetros reflejados en la tabla 7 29.

	x	y	Z
Punto inicial	848.86	232.26	143.54
Punto final	893.47	182.61	933.97
Inicial – Final	-44.61	49.65	-790.43

Tabla 7 29. Distancia total en milímetros del desplazamiento.

Con una distancia calculada de 790 mm y sabiendo que la velocidad del robot puesta por software es de 100 mm/s obtenemos el tiempo total del desplazamiento, figura 7 30.

Distancia (mm)	Velocidad (m/s)	**TIEMPO (s)**
790	100	**7,9**

Tabla 7 30. Tiempo total del movimiento.

Sabiendo el tiempo del desplazamiento y dividiéndolo entre el total de datos, calculamos la frecuencia de envió, tabla 7 31.

Tiempo (s)	Numero de datos	**Frecuencia de envió (s)**
7,9	29	**0,272**

Tabla 7 31. Frecuencia de envió de datos

En la tabla 6.32 se muestra todos los datos enviados al robot una vez procesados, así como la distancia que debe de desplazarse y el tiempo que tarda en realizar dicho desplazamiento.

Dejando enmarcados aquellos que son utilizados por el robot.

Nº	Leídos por el sensor	Enviados al robot	Distancia en (mm) del movimiento a realizar	Tiempo (s) que tarda en ejecutar el movimiento
1	-9867,00	-9867,00	-98,67	0,99
2	-14292,00	-13000,00	-130,00	1,30
3	-15964,00	-13000,00	-130,00	1,30
4	-15634,00	-13000,00	-130,00	1,30
5	-16257,00	-13000,00	-130,00	1,30
6	-17436,00	-13000,00	-130,00	1,30
7	-17431,00	-13000,00	-130,00	1,30
8	-17223,00	-13000,00	-130,00	1,30
9	-18116,00	-13000,00	-130,00	1,30
10	-17468,00	-13000,00	-130,00	1,30
11	-13487,00	-13000,00	-130,00	1,30
12	-14258,00	-13000,00	-130,00	1,30
13	-12944,00	-12944,00	-129,44	1,29
14	-13861,00	-13000,00	-130,00	1,30
15	-12284,00	-12284,00	-122,84	1,23
16	-12611,00	-12611,00	-126,11	1,26
17	-12505,00	-12505,00	-125,05	1,25
18	-11355,00	-11355,00	-113,55	1,14
19	-10135,00	-10135,00	-101,35	1,01
20	-8781,00	-8781,00	-87,81	0,88
21	-9123,00	-9123,00	-91,23	0,91
22	-9534,00	-9534,00	-95,34	0,95
23	-8229,00	-8229,00	-82,29	0,82
24	-7652,00	-7652,00	-76,52	0,77
25	-6948,00	-6948,00	-69,48	0,69
26	-6173,00	-6173,00	-61,73	0,62
27	-8082,00	-8082,00	-80,82	0,81
28	-9095,00	-9095,00	-90,95	0,91
29	-7599,00	-7599,00	-75,99	0,76

Tabla 7 32. Datos enviados, tiempo y distancia realizados por el robot eje Z negativo.

En la figura 7.36 al ser un eje complicado de manejar obtenemos unos valores de fuerza diferentes entre sí. Produciendo muchos saltos en el movimiento y por tanto errores.

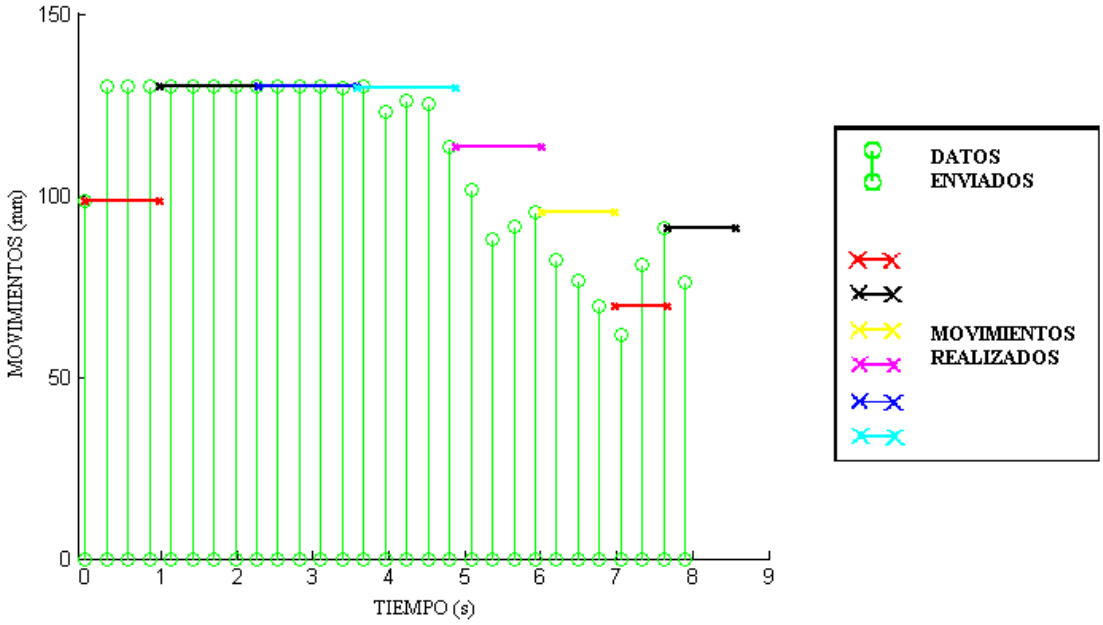


Fig. 7.36. Análisis de los valores de fuerza usados, eje Z negativo.

Se puede comprobar que se ejecuten 8 valores de los 29 que se envían, puesto que no da tiempo a ejecutarlos todos.

El desplazamiento real que realiza el robot, con respecto al de la suma de los datos enviados y ejecutados; es diferente, como se puede apreciar en la tabla 7 33 haciendo por tanto un desplazamiento poco preciso aunque posteriormente se explicara el porqué.

Suma de datos(mm)	Distancia recorrida (mm)
857	790

Tabla 7 33. Desplazamiento real y datos enviados.

7.3. Conclusión del estudio de movimientos.

Se han analizado tres ejes de coordenadas, con el objetivo de conocer una serie de parámetros, que se explicaran a continuación; y que nos ayudarán a comprender mejor los movimientos.

- Frecuencia de envío.

El primero de estos puntos es la frecuencia de envío, que mide el tiempo que se tarda en enviar un dato al robot, para que este después el robot los procese.

Obtenido este dato, se podrá comparar cuánto tarda en recibir un valor el robot y cuanto en realizarlo.

En la tabla 7 34 se observa una frecuencia de envió y un tiempo medio de ejecución de un valor de fuerza, muy similar entre todos los casos. Esta similitud solo es posible si las medidas fueran correctamente tomadas, por eso se descarta el eje Y negativo al tener mucho error.

	Frecuencia de envío (s)	**Tiempo medio de ejecución (s)**
Eje X +	0,342	1,056
Eje x -	0,3985	1,3
Eje y +	0,3566	1,269
Eje Y -	----	----
Eje z +	0,352	1,11
Eje Z -	0,272	1,06
MEDIA DE ROBOT	**0,344**	**1,159**

Tabla 7 34. Frecuencia de envío media y tiempo medio de ejecución de un dato.

Conociendo los valores de la tabla anterior se podría en trabajos futuros o ampliaciones intentar mejorarlos, desde el punto de vista del software o el hardware.

- Porcentaje de desviación.

Este valor nos proporciona el porciento en mm que el robot se ha desviado de los valores que se le han mandado .Pudiendo así saber si se pierden muchos datos en el envío u , el robot comete demasiado error al moverse.

La tabla 7 35 muestra los valores del movimiento teórico y el que el robot realizado de forma práctica, para posteriormente ofrecer el porcentaje de desviación.

	Distancia teórica (mm)	Distancia real (mm)	Porcentaje de desviación (%)
Eje X +	561,1	548	2,34
Eje x -	520	558	7,31
Eje y +	1177	1270	7,91
Eje Y -	----	----	----
Eje z +	441	740	32,12
Eje Z -	857	790	7,82
MEDIA DE ROBOT			**6,347**

Tabla 7 35. Porcentaje en la desviación de la trayectoria.

Observando la tabla anterior al igual que en la tabla 7 34 se desprecia el eje Y negativo por sus errores .Y en este caso concretamente también se desprecia el eje Z positivo puesto que como se puede ver en la figura 6.36 se trata de un movimiento corto y en el que solo se ejecutan unos valores de poco modulo con respecto a los demás repercutiendo posteriormente en la distancia recorrida.

Analizando el porcentaje de error obtenido en la tabla 7 35 se puede concluir en que dado que no se puede interrumpir una ejecución una vez empezada, y teniendo en cuenta todos los ejes podemos considerar que el robot se mueve en tiempo real, al movimiento de la persona.

Al igual que en caso anterior este error también es motivo de mejora para trabajos posteriores.

8. PRESUPUESTO.

El siguiente aparatado contiene el presupuesto aproximado de la realización y puesta en marcha del proyecto descrito.

8.1. Medidas.

8.1.1. Capítulo 1: Material eléctrico y mecánico.

ITEM	Descripción	Cantidad
1	**Sensor fuerza-par JR3.** Modelo 85M35A3.	1
2	**Cable conexión sensor JR3-PC.** Cable con conector RJ-11 de 6 pines.	1
3	**Piezas aluminio para instalación de sensor JR3.** Fabricación de las piezas en aluminio para acoplar el sensor al robot ABB.	12

Tabla 8 1. Medidas – Capítulo 1: Material eléctrico y mecánico.

8.1.2. Capítulo 2: Software y hardware.

ITEM	Descripción	Cantidad
4	**Ordenador sobremesa HP.** Ordenador sobremesa con monitor TFT 17'' y tarjeta de adquisición de datos.	1
5	**SolidEdge.** Incluye programa SolidEdge y licencia correspondiente.	1
6	**Visual Studio 2008.** Incluye programa Visual Studio 2008 y licencia correspondiente.	1
7	**RobotStudio.** Incluye programa RobotStudio con licencia de los paquetes necesarios para la aplicación.	1

Tabla 8 2. Medidas – Capítulo 2: Software y hardware.

8.1.3. Capítulo 3: Programación, diseño y puesta en marcha.

ITEM	Descripción	Cantidad
8	**Diseño de piezas aluminio para instalación del sensor mediante SolidEdge.** Diseño mediante SolidEdge de las piezas para la instalación del sensor en el robot ABB.	15
9	**Programación y desarrollo de interfaz en PC mediante Visual Studio 2008.** Desarrollo de la comunicación PC-SENSOR-ROBOT. Adaptación de la interfaz.	50
10	**Programación de la comunicación y recepción de datos en robot ABB median RobotStudio.** Desarrollo de la aplicación para la extracción de la trama enviada al robot ABB y creación de los movimientos.	30
11	**Montaje e instalación de todos los elementos.** Instalación del sensor JR3 en el robot ABB y puesta en marcha de la aplicación.	20

Tabla 8 3. Medidas – Capítulo 3: Programación, diseño y puesta en marcha.

8.2. Cuadro de precios.

8.2.1. Capítulo 1: Material eléctrico y mecánico.

ITEM	Descripción	Precio
1	**Sensor fuerza-par JR3.** Modelo 85M35A3.	956,68 €/u
2	**Cable conexión sensor JR3-PC.** Cable con conector RJ-11 de 6 pines.	8,56 €/u
3	**Piezas aluminio para instalación de sensor JR3.** Fabricación de las piezas en aluminio para acoplar el sensor al robot ABB.	25 €/h

Tabla 8 4. Cuadro de precios – Capítulo 1: Material eléctrico y mecánico.

8.2.2. Capítulo 2: Software y hardware.

ITEM	Descripción	Precio
4	**Ordenador sobremesa HP.** Ordenador sobremesa con monitor TFT 17'' y tarjeta de adquisición de datos.	1050 €/u
5	**SolidEdge.** Incluye programa SolidEdge y licencia correspondiente.	350 €/u
6	**Visual Studio 2008.** Incluye programa Visual Studio 2008 y licencia correspondiente.	300 €/u
7	**RobotStudio.** Incluye programa RobotStudio con licencia de los paquetes necesarios para la aplicación.	380 €/u

Tabla 8 5. Cuadro de precios – Capítulo 2: Software y hardware.

8.2.3. Capítulo 3: Programación, diseño y puesta en marcha.

ITEM	Descripción	Precio
8	**Diseño de piezas aluminio para instalación del sensor mediante SolidEdge.** Diseño mediante SolidEdge de las piezas para la instalación del sensor en el robot ABB.	27 €/h
9	**Programación y desarrollo de interfaz en PC mediante Visual Studio 2008.** Desarrollo de la comunicación PC-SENSOR-ROBOT. Adaptación de la interfaz.	32 €/h
10	**Programación de la comunicación y recepción de datos en robot ABB median RobotStudio.** Desarrollo de la aplicación para la extracción de la trama enviada al robot ABB y creación de los movimientos.	30 €/h
11	**Montaje e instalación de todos los elementos.** Instalación del sensor JR3 en el robot ABB y puesta en marcha de la aplicación.	20 €/h

Tabla 8 6. Cuadro de precios – Capítulo 3: Programación, diseño y puesta en marcha.

8.3. Presupuesto.

8.3.1. Capítulo 1: Material eléctrico y mecánico.

ITEM	Descripción	Cantidad	Precio	Importe
1	**Sensor fuerza-par JR3.**	1	956,68 €/u	956,68 €
2	**Cable conexión sensor JR3-PC.**	1	8,56 €/u	8,56 €
3	**Piezas aluminio para instalación de sensor JR3.**	12	25 €/h	300 €
	TOTAL CAPÍTULO 1: Material eléctrico y mecánico.			**1265,24 €**

Tabla 8 7. Cuadro de precios – Capítulo 1: Material eléctrico y mecánico.

8.3.2. Capítulo 2: Software y hardware.

ITEM	Descripción	Cantidad	Precio	Importe
4	**Ordenador sobremesa HP.**	1	1050 €/u	1050 €
5	**SolidEdge.**	1	350 €/u	350 €
6	**Visual Studio 2008.**	1	300 €/u	300 €
7	**RobotStudio.**	1	380 €/u	380 €
	TOTAL CAPÍTULO 2: Software y hardware.			**2080,00 €**

Tabla 8 8. Cuadro de precios – Capítulo 2: Software y hardware.

8.3.3. Capítulo 3: Programación, diseño y puesta en marcha.

ITEM	Descripción	Cantidad	Precio	Importe
8	**Diseño de piezas aluminio para instalación del sensor mediante SolidEdge.**	15	27 €/h	405 €
9	**Programación y desarrollo de interfaz en PC mediante Visual Studio 2008.**	50	32 €/h	1600 €
10	**Programación de la comunicación y recepción de datos en robot ABB median RobotStudio.**	30	30 €/h	900 €
11	**Montaje e instalación de todos los elementos.**	20	20 €/h	400 €
	TOTAL CAPÍTULO 3: Programación, diseño y puesta en marcha.			**3305,00 €**

Tabla 8 9. Cuadro de precios – Capítulo 3: Programación, diseño y puesta en marcha.

8.4. Resumen del presupuesto.

CAPÍTULO 1: Material eléctrico y mecánico.	**1265,24 €**
CAPÍTULO 2: Software y hardware.	**2080,00 €**
CAPÍTULO 3: Programación, diseño y puesta en marcha.	**3305, 00 €**
+	
PRESUPUESTO DE EJECUCIÓN MATERIAL.	**6650,24 €**
+	
I.V.A. (18%)	**1197,05 €**
+	
PRESUPUESTO POR LICITACIÓN	**7847, 29 €**

El presupuesto por licitación del presente proyecto asciende a **SIETE MIL OCHOCIENTOS CUARENTA Y SIETE EUROS CON VEINTINUEVE CENTIMOS**.

9. CONCLUSIONES.

Para una mayor comprensión del análisis del proyecto realizado se procederá a la separación de este, en dos partes de acuerdo a su naturaleza.

1. HARDWARE.

Como objetivo primario se pretendía implementar un sistema de fijación que permitiera colocar un sensor de fuerza-par y un soporte a modo de mango del sensor, a un brazo robotizado ABB.

Como sistema elegido se diseñaron dos placas de aluminio, eligiendo este material por su bajo coste, alta maleabilidad y su escaso peso, algo que facilitaba su montaje y no aumentaba en exceso la masa de la punta del robot. Desde el punto de vista del material utilizado no se saca ninguna conclusión negativa que pueda afectar al funcionamiento ni a la vida del proyecto.

Dichas placas son colocadas entre los tres componentes; mango, sensor, robot; a modo de "sándwich" y fijadas con tornillos entre sí. En un principio parecía el mejor sistema tanto por su facilidad como su coste, y lo era si lo que se pretende es controlar fuerzas donde la precisión no sea algo especialmente relevante. Puesto que después de la realización de los ensayos, del punto 7 de la memoria, se descubrió que el mero hecho de fijar en serie todo el sistema, es decir, anclando cada elemento nuevo al anterior mediante tornillos, aunque cumplía con la exigencia de poderse desmontar para dejar el robot libre, se podía aflojar ligeramente algún tornillo, produciendo vibraciones y desajustes en la medida que aunque eran mínimos abría que tenerlos encueta.

El sensor JR3 lleva un mismo cable, para alimentación y datos, haciendo que desde el principio surgiera el problema de su colocación. El brazo robotizado dispone de numerosas articulaciones con varios grados de libertad y los cables de todas ellas, están situados en el interior del robot .Para el proyecto resultaba inviable hacer pasar el cable por el interior del brazo .Por tanto se analizaron diferentes sistemas de fijación y se opto por el de menor coste y que cumpliera con las necesidades del proyecto. La forma fue dejar el cable al descubierto y de forma libre anclado únicamente en la base del robot para que el operario no tuviera ningún contacto. Esta forma de colocación por tanto es válida para este proyecto al no intervenir para nada dentro del radio de acción del robot, ni del usuario.

El problema se encuentra cuando el usuario desea girar el brazo 360 º sobre la base del robot o realiza muchos giros juntos ,puesto que el cable tiende a enrollarse y podría llegar a romperse lo que supondría la pérdida total de datos y el final del movimiento.

Este problema por tanto se queda sin resolución aunque para este proyecto no supone graves alteraciones pudiéndolo clasificar como trabajos futuros.

Para terminar esta parte del análisis cabe remarcar que se han superado los objetivos iniciales de forma genérica ,cumpliendo con las exigencias del proyecto y estudiando que los fallos descubiertos no afectan de una forma significativa al correcto funcionamiento.

2. SOFTWARE

La parte de programación está formada por diferentes objetivos.
El más importante era conseguir una conexión entre un PC que controlaba el sensor JR3 y el robot ABB. Esta comunicación se ha llevado a cabo mediante una conexión TCP/IP formada por sockets.

Para realizar un movimiento son necesarios tres datos, pertenecientes a cada eje cartesiano. La primera dificultad que se presento a la hora de enviar los datos del PC al robot. Puesto que mandándolos de uno en uno retrasaba de forma significativa el movimiento del robot, produciendo una desincronización con respecto a la lectura de los datos.

Llegado a este punto y buscando el mayor rendimiento, se opto por una forma más compleja de transmisión de datos ,consistente en introducir en un solo envió toda la información necesaria para el movimiento ,haciendo por tanto que la decodificación de la información por parte del receptor sea más compleja.

Gracias a esta forma se consiguió la comunicación más rápida entre PC y robot posible. Descubriéndose entonces el principal problema, que hace que el movimiento de la persona y el robot no sea perfectamente síncrono.

La velocidad de procesamiento de la información del PC es muy superior a la del robot. Esto hace que se tenga que retrasar el envió de datos para que el robot le dé tiempo a procesarlos y no se pierdan.

Pese al problema de la desincronización se considera superado el conjunto de objetivos, puesto que aunque siempre es mejorable y posible su optimización, se ha conseguido implementar de forma satisfactoria y correcta la comunicación entre los distintos elementos siendo este su principal objetivo y logro.

También remarcar que se ha conseguido un manejo del robot muy próximo al del ser humano que lo controla.

10. TRABAJOS FUTUROS.

Este proyecto no es sino el primer paso hacia una infinidad de aplicaciones, que tienen como base común la implementación de fuerzas y pares en sistemas robóticos.

Un punto por dónde empezar estaría en la corrección de los diferentes errores explicados en el capítulo 9 de la memoria.

Destacando los errores o mejoras de de hardware se puede intentar diseñar otro soporte de fijación para los tres elementos; mango, sensor y robot; en el que exista un soporte que comprenda de forma fija e inmóvil el mango y el sensor y el único punto de anclaje se encuentre en el robot.

Con respecto al hardware otra mejora que supondría un uso más industrial de este proyecto residiría en crear un sistema para el cable del sensor de fuerza-par JR3 en el que en cada articulación del robot se encuentre fijado el cable suficiente para permitir el máximo movimiento de cada articulación sin producirse tensiones ni posibilidades de ruptura de dicho cable.

Para resolver otros problemas de programación habría que realizar un proyecto para cambiar partes del hardware del robot haciendo que este pudiese procesar más datos lo que haría que no se tuviera que retrasar el envió de los mismos.

Sin tener que cambiar el hardware también se podría mejorar la sincronización del robot y el movimiento que ejerza la persona; teniendo otros factores en cuenta en el movimiento ,como el par o el tiempo de duración de cada elemento de la fuerza.

Durante la programación surgen objetivos nuevos para mejorar el proyecto, pero su complejidad hace que resulte muy difícil su implementación. Este es el caso de una mejora en el interfaz de control del sensor JR3. Consistente en la fusión en un solo programa que permita el control del robot y el sensor, pudiendo así simplificar el capitulo 6, puesta en marcha.

Por último se podría aplicar este sistema a brazos robóticos que tuvieran que ejercer no solo fuerzas, sino también pares como por ejemplo brazos que tuvieran que colocar tornillos con un determinado apriete y ángulo de colocación.

11. BIBLIOGRAFIA.

[1] http://www.slideshare.net/mmastino/geometra-de-los-robots (consultada el 01/01/2011)

[2] http://muatrip.blogspot.com/2011_05_01_archive.html (consultada el 01/01/2011)

[3] http://cfievalladolid2.net/tecno/cyr_01/robotica/industrial.htm (consultada el 01/01/2011)

[4] http://www.abb.es/ (consultada el 01/01/2011)

[5] http://www.profesorblanco.com.ar/apuntesteco/SENSORES%201.pdf (consultada el 01/01/2011)

[6] http://www.isa.cie.uva.es/~maria/sensores.pdf (consultada el 01/01/2011)

[7] http://www.guemisa.com/articul/html/sensores.htm (consultada el 01/01/2011)

[8] http://www.abb.es/product/seitp327/df90f6fe2c1ffc64c125725100252d4d.aspx

[9] http://msdn.microsoft.com/es-es/library/aa187919.aspx

[10] http://www.solarsockets.solar-opensource.com/index.php/SolarSockets

[11]http://es.scribd.com/doc/39792915/Manual-Solid-Edge

[12]http://www.ehu.es/asignaturasKO/DibujoInd/Solid_Edge_v16.pdf

12. ANEXOS.

12.1. Archivos .CPP

12.1.1. DlgProxy.cpp

```
// DlgProxy.cpp : implementation file
//

#include "stdafx.h"
#include "jr3pci.h"
#include "DlgProxy.h"
#include "jr3pciDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CJr3pciDlgAutoProxy

IMPLEMENT_DYNCREATE(CJr3pciDlgAutoProxy, CCmdTarget)

CJr3pciDlgAutoProxy::CJr3pciDlgAutoProxy()
{
	EnableAutomation();

	// To keep the application running as long as an automation
	//	object is active, the constructor calls AfxOleLockApp.
	AfxOleLockApp();

	// Get access to the dialog through the application's
	//  main window pointer.  Set the proxy's internal pointer
	//  to point to the dialog, and set the dialog's back pointer to
	//  this proxy.
	ASSERT (AfxGetApp()->m_pMainWnd != NULL);
	ASSERT_VALID (AfxGetApp()->m_pMainWnd);
	ASSERT_KINDOF(CJr3pciDlg, AfxGetApp()->m_pMainWnd);
	m_pDialog = (CJr3pciDlg*) AfxGetApp()->m_pMainWnd;
	m_pDialog->m_pAutoProxy = this;
}

CJr3pciDlgAutoProxy::~CJr3pciDlgAutoProxy()
{
	// To terminate the application when all objects created with
	//	with automation, the destructor calls AfxOleUnlockApp.
	//  Among other things, this will destroy the main dialog
	if (m_pDialog != NULL)
		m_pDialog->m_pAutoProxy = NULL;
	AfxOleUnlockApp();
}
```

```
void CJr3pciDlgAutoProxy::OnFinalRelease()
{
	// When the last reference for an automation object is released
	// OnFinalRelease is called.  The base class will automatically
	// deletes the object.  Add additional cleanup required for your
	// object before calling the base class.

	CCmdTarget::OnFinalRelease();
}

BEGIN_MESSAGE_MAP(CJr3pciDlgAutoProxy, CCmdTarget)
	//{{AFX_MSG_MAP(CJr3pciDlgAutoProxy)
		// NOTE - the ClassWizard will add and remove mapping macros here.
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

BEGIN_DISPATCH_MAP(CJr3pciDlgAutoProxy, CCmdTarget)
	//{{AFX_DISPATCH_MAP(CJr3pciDlgAutoProxy)
		// NOTE - the ClassWizard will add and remove mapping macros here.
	//}}AFX_DISPATCH_MAP
END_DISPATCH_MAP()

// Note: we add support for IID_IJr3pci to support typesafe binding
//  from VBA.  This IID must match the GUID that is attached to the
//  dispinterface in the .ODL file.

// {A502BA66-101F-49A8-802F-1FDD0089A192}
static const IID IID_IJr3pci =
{ 0xa502ba66, 0x101f, 0x49a8, { 0x80, 0x2f, 0x1f, 0xdd, 0x0, 0x89, 0xa1, 0x92 } };

BEGIN_INTERFACE_MAP(CJr3pciDlgAutoProxy, CCmdTarget)
	INTERFACE_PART(CJr3pciDlgAutoProxy, IID_IJr3pci, Dispatch)
END_INTERFACE_MAP()

// The IMPLEMENT_OLECREATE2 macro is defined in StdAfx.h of this project
// {85A6144F-92D1-457A-A958-D32D7D417890}
IMPLEMENT_OLECREATE2(CJr3pciDlgAutoProxy, "Jr3pci.Application", 0x85a6144f, 0x92d1, 0x457a,
0xa9, 0x58, 0xd3, 0x2d, 0x7d, 0x41, 0x78, 0x90)

/////////////////////////////////////////////////////////////////////////////
	// CJr3pciDlgAutoProxy message handlers
```

12.1.2. Jr3pci.cpp

```
// jr3pci.cpp : Defines the class behaviors for the application.
//

#include "stdafx.h"
#include "jr3pci.h"
#include "jr3pciDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CJr3pciApp

BEGIN_MESSAGE_MAP(CJr3pciApp, CWinApp)
	//{{AFX_MSG_MAP(CJr3pciApp)
		// NOTE - the ClassWizard will add and remove mapping macros here.
		//    DO NOT EDIT what you see in these blocks of generated code!
	//}}AFX_MSG
	ON_COMMAND(ID_HELP, CWinApp::OnHelp)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CJr3pciApp construction

CJr3pciApp::CJr3pciApp()
{
	// TODO: add construction code here,
	// Place all significant initialization in InitInstance
}

/////////////////////////////////////////////////////////////////////////////
// The one and only CJr3pciApp object

CJr3pciApp theApp;

/////////////////////////////////////////////////////////////////////////////
// CJr3pciApp initialization

BOOL CJr3pciApp::InitInstance()
{
	// Initialize OLE libraries
	if (!AfxOleInit())
	{
		AfxMessageBox(IDP_OLE_INIT_FAILED);
		return FALSE;
	}

	AfxEnableControlContainer();

	// Standard initialization
	// If you are not using these features and wish to reduce the size
```

```
	//  of your final executable, you should remove from the following
	//  the specific initialization routines you do not need.

#ifdef _AFXDLL
	Enable3dControls();			// Call this when using MFC in a shared DLL
#else
	Enable3dControlsStatic();	// Call this when linking to MFC statically
#endif

	// Parse the command line to see if launched as OLE server
	if (RunEmbedded() || RunAutomated())
	{
		// Register all OLE server (factories) as running.  This enables the
		//  OLE libraries to create objects from other applications.
		COleTemplateServer::RegisterAll();
	}
	else
	{
		// When a server application is launched stand-alone, it is a good idea
		//  to update the system registry in case it has been damaged.
		COleObjectFactory::UpdateRegistryAll();
	}

	CJr3pciDlg dlg;
	m_pMainWnd = &dlg;
	int nResponse = dlg.DoModal();
	if (nResponse == IDOK)
	{
		// TODO: Place code here to handle when the dialog is
		//  dismissed with OK
	}
	else if (nResponse == IDCANCEL)
	{
		// TODO: Place code here to handle when the dialog is
		//  dismissed with Cancel
	}

	// Since the dialog has been closed, return FALSE so that we exit the
	//  application, rather than start the application's message pump.
	return FALSE;
}
```

12.1.3. Jr3pciDlg.cpp

```
// jr3pciDlg.cpp : implementation file
//

#include "stdafx.h"
#include "jr3pci.h"
#include "jr3pciDlg.h"
#include "DlgProxy.h"
#include "jr3pci_ft.h"
//Ficheros cabezera de SolarSockets necesarios para la comunicación sockets (TCP/IP)
#include <SolarSockets/SolarSockets++.h>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <sstream>
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
short initiated=0;
UINT timer_ID;
CString mmsg;
DWORD vendor_ID, device_ID;
short pnum,nump,bnum;
//Declaraciones de las variables necesarias para la nueva aplicación.
string datofx="",datofy="",datofz="",datomx="",datomy="",datomz="";
string stringlongitudfx="",stringlongitudfy="",stringlongitudfz="";
string datocompleto="";
char charlongitudfx[1],charlongitudfy[1],charlongitudfz[1];
short primeravez=1;
int contador=0;
int nTempx=0;
int nTempy=0;
int nTempz=0;
int longitudfx=0;
int longitudfy=0;
int longitudfz=0;
int maximo=13000;
int minimo=150;
int almacenardatos=0;


//Creación del fichero para almacenar los datos leidos.
FILE * pFile= fopen ("datos_sensor.txt","w");


ClienteTest Cliente;
ClienteTest::ClienteTest(void)
{
}

ClienteTest::~ClienteTest(void)
{
}
```

```
class CAboutDlg : public CDialog
{
public:
	CAboutDlg();


	enum { IDD = IDD_ABOUTBOX };

	protected:
	virtual void DoDataExchange(CDataExchange* pDX);


protected:

	DECLARE_MESSAGE_MAP()
};




CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{

}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);

}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)

END_MESSAGE_MAP()



IMPLEMENT_DYNAMIC(CJr3pciDlg, CDialog);

CJr3pciDlg::CJr3pciDlg(CWnd* pParent /*=NULL*/)
	: CDialog(CJr3pciDlg::IDD, pParent)
{

	m_hIcon = AfxGetApp()->LoadIcon(IDR_jr3pci);
	m_hIcon1 = AfxGetApp()->LoadIcon(IDI_read);
	m_pAutoProxy = NULL;
}

CJr3pciDlg::~CJr3pciDlg()
{

	if (m_pAutoProxy != NULL)
		m_pAutoProxy->m_pDialog = NULL;
}
```

```
void CJr3pciDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	DDX_Control(pDX, IDC_bnum, m_bnum);
	DDX_Control(pDX, IDC_vect, m_vect);
	DDX_Control(pDX, IDC_about, m_about);
	DDX_Control(pDX, IDC_pnumcombo, m_pnum);
	DDX_Control(pDX, IDC_mz4, m_mz4);
	DDX_Control(pDX, IDC_mz3, m_mz3);
	DDX_Control(pDX, IDC_v8, m_v8);
	DDX_Control(pDX, IDC_v7, m_v7);
	DDX_Control(pDX, IDC_v6, m_v6);
	DDX_Control(pDX, IDC_v5, m_v5);
	DDX_Control(pDX, IDC_v4, m_v4);
	DDX_Control(pDX, IDC_v3, m_v3);
	DDX_Control(pDX, IDC_PROGRESS_v8, m_pv8);
	DDX_Control(pDX, IDC_PROGRESS_v7, m_pv7);
	DDX_Control(pDX, IDC_PROGRESS_v6, m_pv6);
	DDX_Control(pDX, IDC_PROGRESS_v5, m_pv5);
	DDX_Control(pDX, IDC_PROGRESS_v4, m_pv4);
	DDX_Control(pDX, IDC_PROGRESS_v3, m_pv3);
	DDX_Control(pDX, IDC_PROGRESS_mz4, m_pmz4);
	DDX_Control(pDX, IDC_PROGRESS_mz3, m_pmz3);
	DDX_Control(pDX, IDC_PROGRESS_mz2, m_pmz2);
	DDX_Control(pDX, IDC_PROGRESS_my4, m_pmy4);
	DDX_Control(pDX, IDC_PROGRESS_my3, m_pmy3);
	DDX_Control(pDX, IDC_PROGRESS_my2, m_pmy2);
	DDX_Control(pDX, IDC_PROGRESS_mx4, m_pmx4);
	DDX_Control(pDX, IDC_PROGRESS_mx3, m_pmx3);
	DDX_Control(pDX, IDC_PROGRESS_mx2, m_pmx2);
	DDX_Control(pDX, IDC_PROGRESS_fz4, m_pfz4);
	DDX_Control(pDX, IDC_PROGRESS_fz3, m_pfz3);
	DDX_Control(pDX, IDC_PROGRESS_fz2, m_pfz2);
	DDX_Control(pDX, IDC_PROGRESS_fy4, m_pfy4);
	DDX_Control(pDX, IDC_PROGRESS_fy3, m_pfy3);
	DDX_Control(pDX, IDC_PROGRESS_fy2, m_pfy2);
	DDX_Control(pDX, IDC_PROGRESS_fx4, m_pfx4);
	DDX_Control(pDX, IDC_PROGRESS_fx3, m_pfx3);
	DDX_Control(pDX, IDC_PROGRESS_fx2, m_pfx2);
	DDX_Control(pDX, IDC_mz2, m_mz2);
	DDX_Control(pDX, IDC_my4, m_my4);
	DDX_Control(pDX, IDC_my3, m_my3);
	DDX_Control(pDX, IDC_my2, m_my2);
	DDX_Control(pDX, IDC_mx4, m_mx4);
	DDX_Control(pDX, IDC_mx3, m_mx3);
	DDX_Control(pDX, IDC_mx2, m_mx2);
	DDX_Control(pDX, IDC_fz4, m_fz4);
	DDX_Control(pDX, IDC_fz3, m_fz3);
	DDX_Control(pDX, IDC_fz2, m_fz2);
	DDX_Control(pDX, IDC_fy4, m_fy4);
	DDX_Control(pDX, IDC_fy3, m_fy3);
	DDX_Control(pDX, IDC_fy2, m_fy2);
	DDX_Control(pDX, IDC_fx4, m_fx4);
	DDX_Control(pDX, IDC_fx3, m_fx3);
	DDX_Control(pDX, IDC_fx2, m_fx2);
```

```
	DDX_Control(pDX, IDC_nump, m_nump);
	DDX_Control(pDX, IDC_did, m_did);
	DDX_Control(pDX, IDC_vid, m_vid);
	DDX_Control(pDX, IDC_offnum, m_offnum);
	DDX_Control(pDX, IDC_msg, m_msg);
	DDX_Control(pDX, IDC_setoff, m_setoff);
	DDX_Control(pDX, IDC_rminfs, m_rminfs);
	DDX_Control(pDX, IDC_rMfs, m_rmfs);
	DDX_Control(pDX, IDC_info, m_info);
	DDX_Control(pDX, IDC_detFS, m_setfs);
	DDX_Control(pDX, IDOK, m_ok);
	DDX_Control(pDX, IDC_read, m_read);
	DDX_Control(pDX, IDC_stop, m_stop);
	DDX_Control(pDX, IDC_v2, m_v2);
	DDX_Control(pDX, IDC_v1, m_v1);
	DDX_Control(pDX, IDC_units, m_units);
	DDX_Control(pDX, IDC_thickness, m_thickness);
	DDX_Control(pDX, IDC_software, m_software);
	DDX_Control(pDX, IDC_serial, m_serial);
	DDX_Control(pDX, IDC_roff, m_roff);
	DDX_Control(pDX, IDC_PROGRESS_v2, m_pv2);
	DDX_Control(pDX, IDC_PROGRESS_v1, m_pv1);
	DDX_Control(pDX, IDC_PROGRESS_mz, m_pmz);
	DDX_Control(pDX, IDC_PROGRESS_my, m_pmy);
	DDX_Control(pDX, IDC_PROGRESS_mx, m_pmx);
	DDX_Control(pDX, IDC_PROGRESS_fz, m_pfz);
	DDX_Control(pDX, IDC_PROGRESS_fy, m_pfy);
	DDX_Control(pDX, IDC_PROGRESS_fx, m_pfx);
	DDX_Control(pDX, IDC_off_mz, m_off_mz);
	DDX_Control(pDX, IDC_off_my, m_off_my);
	DDX_Control(pDX, IDC_off_mx, m_off_mx);
	DDX_Control(pDX, IDC_off_fz, m_off_fz);
	DDX_Control(pDX, IDC_off_fy, m_off_fy);
	DDX_Control(pDX, IDC_off_fx, m_off_fx);
	DDX_Control(pDX, IDC_mz, m_mz);
	DDX_Control(pDX, IDC_my, m_my);
	DDX_Control(pDX, IDC_mx, m_mx);
	DDX_Control(pDX, IDC_model, m_model);
	DDX_Control(pDX, IDC_init, m_init);
	DDX_Control(pDX, IDC_fz, m_fz);
	DDX_Control(pDX, IDC_fy, m_fy);
	DDX_Control(pDX, IDC_fx, m_fx);
	DDX_Control(pDX, IDC_fs_mz, m_fsmz);
	DDX_Control(pDX, IDC_fs_my, m_fsmy);
	DDX_Control(pDX, IDC_fs_mx, m_fsmx);
	DDX_Control(pDX, IDC_fs_fz, m_fsfz);
	DDX_Control(pDX, IDC_fs_fy, m_fsfy);
	DDX_Control(pDX, IDC_fs_fx, m_fsfx);
	DDX_Control(pDX, IDC_eeprom, m_eeprom);
	DDX_Control(pDX, IDC_adc, m_adc);
}

BEGIN_MESSAGE_MAP(CJr3pciDlg, CDialog)

	ON_WM_SYSCOMMAND()
	ON_WM_PAINT()
```

```
	ON_WM_QUERYDRAGICON()
	ON_WM_CLOSE()
	ON_BN_CLICKED(IDC_init, Oninit)
	ON_BN_CLICKED(IDC_roff, Onroff)
	ON_BN_CLICKED(IDC_read, Onread)
	ON_BN_CLICKED(IDC_stop, Onstop)
	ON_WM_TIMER()
	ON_BN_CLICKED(IDC_detFS, OndetFS)
	ON_BN_CLICKED(IDC_rMfs, OnrMfs)
	ON_BN_CLICKED(IDC_rminfs, Onrminfs)
	ON_BN_CLICKED(IDC_setoff, Onsetoff)
	ON_BN_CLICKED(IDC_info, Oninfo)
	ON_CBN_DROPDOWN(IDC_pnumcombo, OnDropdownpnumcombo)
	ON_CBN_SELENDOK(IDC_pnumcombo, OnSelendokpnumcombo)
	ON_CBN_SELENDCANCEL(IDC_pnumcombo, OnSelendcancelpnumcombo)
	ON_BN_CLICKED(IDC_about, Onabout)
	ON_STN_CLICKED(IDC_mx3, &CJr3pciDlg::OnStnClickedmx3)
END_MESSAGE_MAP()


BOOL CJr3pciDlg::OnInitDialog()
{
	CDialog::OnInitDialog();

	ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
	ASSERT(IDM_ABOUTBOX < 0xF000);

	CMenu* pSysMenu = GetSystemMenu(FALSE);
	if (pSysMenu != NULL)
	{
		CString strAboutMenu;
		strAboutMenu.LoadString(IDS_ABOUTBOX);
		if (!strAboutMenu.IsEmpty())
		{
			pSysMenu->AppendMenu(MF_SEPARATOR);
			pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
		}
	}


	SetIcon(m_hIcon, TRUE);
	SetIcon(m_hIcon, FALSE);


	m_init.SetIcon(m_hIcon1);
	m_read.SetIcon(AfxGetApp()->LoadIcon(IDI_start));
  m_stop.SetIcon(AfxGetApp()->LoadIcon(IDI_stop));
	m_init.SetIcon(AfxGetApp()->LoadIcon(IDI_init));
	m_ok.SetIcon(AfxGetApp()->LoadIcon(IDI_out));
	m_about.SetIcon(AfxGetApp()->LoadIcon(IDI_info));
	mmsg.Format("%x",0x1762);
	m_vid.SetWindowText(mmsg);
	mmsg.Format("%x",0x5112);
	m_did.SetWindowText(mmsg);
	m_msg.SetWindowText("Listed IDs and Board Number are the ones by DEFAULT. Change
them if needed.");
```

```
	m_bnum.SetWindowText("1");



	{

		m_tooltip.Create(this);
		m_tooltip.Activate(TRUE);

		m_tooltip.AddTool(GetDlgItem(IDC_init), "Download DSP CODE and Start
Sensor.");
		m_tooltip.AddTool(GetDlgItem(IDOK), "Bye!");
		m_tooltip.AddTool(GetDlgItem(IDC_read), "Read FT Data: 1ms");
		m_tooltip.AddTool(GetDlgItem(IDC_stop), "Stop reading sensor.");
	m_tooltip.AddTool(GetDlgItem(IDC_roff), "Reset Offsets");
	m_tooltip.AddTool(GetDlgItem(IDC_setoff), "Set Offsets: write-in the values");
		m_tooltip.AddTool(GetDlgItem(IDC_detFS), "Set FScales: write-in the values");
		m_tooltip.AddTool(GetDlgItem(IDC_rMfs), "Read Recommended Maximum FS");
		m_tooltip.AddTool(GetDlgItem(IDC_rminfs), "Read Recommended Minimum FS");



	}
	return TRUE;
}

void CJr3pciDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
	if ((nID & 0xFFF0) == IDM_ABOUTBOX)
	{
		CAboutDlg dlgAbout;
		dlgAbout.DoModal();
	}
	else
	{
		CDialog::OnSysCommand(nID, lParam);
	}
}



void CJr3pciDlg::OnPaint()
{
	if (IsIconic())
	{
		CPaintDC dc(this);

		SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);

		int cxIcon = GetSystemMetrics(SM_CXICON);
		int cyIcon = GetSystemMetrics(SM_CYICON);
		CRect rect;
		GetClientRect(&rect);
		int x = (rect.Width() - cxIcon + 1) / 2;
		int y = (rect.Height() - cyIcon + 1) / 2;
```

```
		dc.DrawIcon(x, y, m_hIcon);
	}
	else
	{
		CDialog::OnPaint();
	}
}


HCURSOR CJr3pciDlg::OnQueryDragIcon()
{
	return (HCURSOR) m_hIcon;
}



void CJr3pciDlg::OnClose()
{
	if (CanExit())
		CDialog::OnClose();
}

void CJr3pciDlg::OnOK()
{
	if (CanExit())
		CDialog::OnOK();
}

void CJr3pciDlg::OnCancel()
{
	if (CanExit())
		CDialog::OnCancel();
}

BOOL CJr3pciDlg::CanExit()
{

	if (m_pAutoProxy != NULL)
	{
		ShowWindow(SW_HIDE);


		return FALSE;



	}

	return TRUE;
}

void CJr3pciDlg::Oninit()
{
	CString msg,msg1;
```

```
        int a;
        DWORD os_version;



        if (!initiated)
        {
         m_bnum.GetWindowText(msg);
         bnum=(short)strtoul(msg,NULL,10);

   m_vid.GetWindowText(msg);
         vendor_ID=strtoul(msg,NULL,16);
   m_did.GetWindowText(msg);
         device_ID=strtoul(msg,NULL,16);
         strtoul(msg,NULL,16);
         msg1=msg.Right(1);
         m_nump.SetWindowText(msg1);
         nump=(short)strtoul(msg1,NULL,10);
   os_version=GetVersion();
 //Creación de las columnas en el fichero para almacenar los datos guardados.
        fputs ("  Fx                    Fy                    Fz
                  Mx                    My                    Mz",pFile);
                fputs ("\n",pFile);
 //Conexíón del PC con el robot ABB. Espera de 100 ms para establecer la conexión correctamente.
Cliente.Connect("192.168.133.2", 4410);
Sleep(100);



         a=init_jr3(vendor_ID, device_ID, bnum,nump,1);
         m_pnum.SetWindowText("0");
         if (a>=0) m_msg.SetWindowText("DSP Code was downloded OK"); else
         {
                if (a==-96)
                msg.Format("Download Error. %d",a);
                if (a==-92)
                msg.Format("Windriver version error: %d",a);
                if (a==-93)
                msg.Format("PCI Card Not Found: %d",a);
                if (a==-94)
                msg.Format("Card Not In Range: %d",a);
                if (a==-95)
                msg.Format("Failed Locking PCI Card (already in use): %d",a);
                if (a==-91)

                        msg.Format("Failled to open Handle to Windriver ... run wdreg %d",a);

                if (a==-90)
                  msg.Format("IDM file with DSP code not found.");


                AfxMessageBox(msg);
                return;
         }
```

```
            initiated=1;

            m_ok.EnableWindow(FALSE);
            m_read.EnableWindow(TRUE);
            m_stop.EnableWindow(TRUE);
            m_info.EnableWindow(TRUE);
            m_setfs.EnableWindow(TRUE);
            m_setoff.EnableWindow(TRUE);
            m_roff.EnableWindow(TRUE);
            m_rminfs.EnableWindow(TRUE);
            m_rmfs.EnableWindow(TRUE);
            m_pnum.EnableWindow(TRUE);

            m_pnum.SetWindowText("0");
    pnum=0;



          } else
          {

            close_jr3();
            initiated=0;
            m_ok.EnableWindow(TRUE);
            m_read.EnableWindow(FALSE);
            m_stop.EnableWindow(FALSE);
            m_info.EnableWindow(FALSE);
            m_setfs.EnableWindow(FALSE);
            m_setoff.EnableWindow(FALSE);
            m_roff.EnableWindow(FALSE);
            m_rminfs.EnableWindow(FALSE);
            m_rmfs.EnableWindow(FALSE);
            m_msg.SetWindowText("Bye for NOW.");
            m_pnum.EnableWindow(FALSE);

          }

}

void CJr3pciDlg::Onroff()
{

          CString msg;
   m_offnum.GetWindowText(msg);
          m_pnum.GetWindowText(msg);
          pnum=(short)strtoul(msg,NULL,10);
          reset_offsets(pnum);
          m_msg.SetWindowText("Offsets reseted to current FT data");
   CJr3pciDlg::Oninfo();


}

void CJr3pciDlg::Onread()
{
```

```
struct force_array off;
        short range;
        CString msg;

        m_pnum.GetWindowText(msg);
        pnum=(short)strtoul(msg,NULL,10);
        msg.Format("%d",pnum);
        m_pnum.SetWindowText(msg);

        m_init.EnableWindow(FALSE);


        range = 16384;

        m_pfx.SetRange(0,range);
        m_pfy.SetRange(0,range);
        m_pfz.SetRange(0,range);
        m_pmx.SetRange(0,range);
        m_pmy.SetRange(0,range);
        m_pmz.SetRange(0,range);
        m_pv1.SetRange(0,range);
        m_pv2.SetRange(0,range);

        m_pfx2.SetRange(0,range);
        m_pfy2.SetRange(0,range);
        m_pfz2.SetRange(0,range);
        m_pmx2.SetRange(0,range);
        m_pmy2.SetRange(0,range);
        m_pmz2.SetRange(0,range);
        m_pv3.SetRange(0,range);
        m_pv4.SetRange(0,range);

        m_pfx3.SetRange(0,range);
        m_pfy3.SetRange(0,range);
        m_pfz3.SetRange(0,range);
        m_pmx3.SetRange(0,range);
        m_pmy3.SetRange(0,range);
        m_pmz3.SetRange(0,range);
        m_pv5.SetRange(0,range);
        m_pv6.SetRange(0,range);

        m_pfx4.SetRange(0,range);
        m_pfy4.SetRange(0,range);
        m_pfz4.SetRange(0,range);
        m_pmx4.SetRange(0,range);
        m_pmy4.SetRange(0,range);
        m_pmz4.SetRange(0,range);
        m_pv7.SetRange(0,range);
        m_pv8.SetRange(0,range);

        off = read_ftdata(FILTER2,0);

msg.Format("%d",(off.fx));
        m_fx.SetWindowText(msg);
        m_pfx.SetPos(abs(off.fx));
```

```
msg.Format("%d",off.fy);
        m_fy.SetWindowText(msg);
        m_pfy.SetPos(abs(off.fy));
msg.Format("%d", (off.fz));
        m_fz.SetWindowText(msg);
        m_pfz.SetPos(abs(off.fz));

        msg.Format("%d",off.mx);
        m_mx.SetWindowText(msg);
        m_pmx.SetPos(abs(off.mx));

        msg.Format("%d",off.my);
        m_my.SetWindowText(msg);
        m_pmy.SetPos(abs(off.my));

msg.Format("%d",off.mz);
        m_mz.SetWindowText(msg);
        m_pmz.SetPos(abs(off.mz));

msg.Format("%d",off.v1);
        m_v1.SetWindowText(msg);
        m_pv1.SetPos(abs(off.v1));
msg.Format("%d",off.v2);
        m_v2.SetWindowText(msg);
        m_pv2.SetPos(abs(off.v2));

timer_ID=SetTimer(NULL,1, NULL);

        msg.Format("%d",off.fx);
        m_fx2.SetWindowText(msg);
        m_pfx2.SetPos(abs(off.fx));
msg.Format("%d",off.fy);
        m_fy2.SetWindowText(msg);
        m_pfy2.SetPos(abs(off.fy));
msg.Format("%d",off.fz);
        m_fz2.SetWindowText(msg);
        m_pfz2.SetPos(abs(off.fz));
msg.Format("%d",off.mx);
        m_mx2.SetWindowText(msg);
        m_pmx2.SetPos(abs(off.mx));

msg.Format("%d",off.my);
        m_my2.SetWindowText(msg);
        m_pmy2.SetPos(abs(off.my));

        msg.Format("%d",off.mz);
        m_mz2.SetWindowText(msg);
        m_pmz2.SetPos(abs(off.mz));

msg.Format("%d",off.v1);
        m_v3.SetWindowText(msg);
        m_pv3.SetPos(abs(off.v1));
msg.Format("%d",off.v2);
        m_v4.SetWindowText(msg);
        m_pv4.SetPos(abs(off.v2));
```

```
        msg.Format("%d",off.fx);
        m_fx3.SetWindowText(msg);
        m_pfx3.SetPos(abs(off.fx));
msg.Format("%d",off.fy);
        m_fy3.SetWindowText(msg);
        m_pfy3.SetPos(abs(off.fy));
msg.Format("%d",off.fz);
        m_fz3.SetWindowText(msg);
        m_pfz3.SetPos(abs(off.fz));
msg.Format("%d",off.mx);
        m_mx3.SetWindowText(msg);
        m_pmx3.SetPos(abs(off.mx));

msg.Format("%d",off.my);
        m_my3.SetWindowText(msg);
        m_pmy3.SetPos(abs(off.my));

        msg.Format("%d",off.mz);
        m_mz3.SetWindowText(msg);
        m_pmz3.SetPos(abs(off.mz));

msg.Format("%d",off.v1);
        m_v5.SetWindowText(msg);
        m_pv5.SetPos(abs(off.v1));
msg.Format("%d",off.v2);
        m_v6.SetWindowText(msg);
        m_pv6.SetPos(abs(off.v2));


        msg.Format("%d",off.fx);
        m_fx4.SetWindowText(msg);
        m_pfx4.SetPos(abs(off.fx));
msg.Format("%d",off.fy);
        m_fy4.SetWindowText(msg);
        m_pfy4.SetPos(abs(off.fy));
msg.Format("%d",off.fz);
        m_fz4.SetWindowText(msg);
        m_pfz4.SetPos(abs(off.fz));
msg.Format("%d",off.mx);
        m_mx4.SetWindowText(msg);
        m_pmx4.SetPos(abs(off.mx));

msg.Format("%d",off.my);
        m_my4.SetWindowText(msg);
        m_pmy4.SetPos(abs(off.my));

        msg.Format("%d",off.mz);
        m_mz4.SetWindowText(msg);
        m_pmz4.SetPos(abs(off.mz));

msg.Format("%d",off.v1);
        m_v7.SetWindowText(msg);
        m_pv7.SetPos(abs(off.v1));
msg.Format("%d",off.v2);
        m_v8.SetWindowText(msg);
        m_pv8.SetPos(abs(off.v2));
```

```
   m_msg.SetWindowText("Reading FT data at 1 ms ...");


}

void CJr3pciDlg::Onstop()
{

          KillTimer(timer_ID);

   m_msg.SetWindowText("Timer killed ... ");
          m_init.EnableWindow(TRUE);

}

void CJr3pciDlg::OnTimer(UINT nIDEvent)
{
   struct force_array off;
          short i;
          CString msg;
          timer_ID = nIDEvent;


   for (i=0;i<=nump-1;i++)
          {

          off = read_ftdata(FILTER2,i);


          // Processor 1
   if (i == 0)
          {

    msg.Format("%d",off.fx);
           m_fx.SetWindowText(msg);
           //Estracción de los valores de fuerza. Conversión mediante atoi a un entero para su
comparación.
           nTempx = atoi (msg);
           datofx = msg;
           m_pfx.SetPos(abs(off.fx));

    msg.Format("%d",off.fy);
           m_fy.SetWindowText(msg);
           nTempy = atoi (msg);
           datofy = msg;
           m_pfy.SetPos(abs(off.fy));

    msg.Format("%d",off.fz);
           m_fz.SetWindowText(msg);
    nTempz = atoi (msg);
           datofz = msg;
```

```
            m_pfz.SetPos(abs(off.fz));

    msg.Format("%d",off.mx);
            m_mx.SetWindowText(msg);
            datomx=msg;
            m_pmx.SetPos(abs(off.mx));

    msg.Format("%d",off.my);
            m_my.SetWindowText(msg);
            datomy=msg;
            m_pmy.SetPos(abs(off.my));

            msg.Format("%d",off.mz);
            m_mz.SetWindowText(msg);
            datomz=msg;
            m_pmz.SetPos(abs(off.mz));

    msg.Format("%d",off.v1);
            m_v1.SetWindowText(msg);

            m_pv1.SetPos(abs(off.v1));
    msg.Format("%d",off.v2);
            m_v2.SetWindowText(msg);
            m_pv2.SetPos(abs(off.v2));

            //Almacenar datos en archivo txt.
if (almacenardatos==1)
{
          fputs ("\n",pFile);
          fwrite (&datofx,sizeof(datofx),1,pFile);
          fwrite (&datofy,sizeof(datofy),1,pFile);
          fwrite (&datofz,sizeof(datofz),1,pFile);
          fwrite (&datomx,sizeof(datomx),1,pFile);
          fwrite (&datomy,sizeof(datomy),1,pFile);
          fwrite (&datomz,sizeof(datomz),1,pFile);

}

//Comprobación de los valores de fuerza para ver si se encuentran dentro del rango establecido.
if ((nTempx>-minimo) && (nTempx<minimo) && (nTempy>-minimo) && (nTempy<minimo) &&
(nTempz>-minimo) && (nTempz<minimo))
{
datofx="";
datofy="";
datofz="";

nTempx=0;
nTempy=0;
nTempz=0;


}



 if (nTempx<-maximo)
```

```
{
datofx="-13000";
}
else if (nTempx>maximo)

{
datofx="13000";
}
else if (nTempy<-maximo)

{
datofy="-13000";
}
else if (nTempy>maximo)

{
datofy="13000";
}
else if (nTempz<-maximo)

{
datofz="-13000";
}
else if (nTempz>maximo)

{
datofz="13000";
}



// Condición despues de conectar para evitar enviar los primeros datos
// medidos por el sensor, los cuales son erroneos.
if (primeravez==1)
{
        for (contador=0;contador<=4;contador++)
        {

    datocompleto="";
        stringlongitudfx="";
            stringlongitudfy="";
        stringlongitudfz="";
            datofx = "";
            datofy = "";
            datofz = "";
            Sleep(500);
        Cliente.Send(datocompleto);
    primeravez=0;

        }
}

//Medida de la longitud de los valores de fuerzas para formar la trama.
longitudfx=strlen(datofx.c_str());
itoa (longitudfx,charlongitudfx,10);
```

```
stringlongitudfx=charlongitudfx;
stringlongitudfx.resize (1);
datocompleto+=stringlongitudfx;


longitudfy=strlen(datofy.c_str());
itoa (longitudfy,charlongitudfy,10);
stringlongitudfy=charlongitudfy;
stringlongitudfy.resize (1);
datocompleto+=stringlongitudfy;


longitudfz=strlen(datofz.c_str());
itoa (longitudfz,charlongitudfz,10);
stringlongitudfz=charlongitudfz;
stringlongitudfz.resize (1);
datocompleto+=stringlongitudfz;

//Se completa la trama con los valores de las tres fuerzas.
// TRAMA COMPLETA = [longitud de fuerza X + longitud de fuerza Y + longitud de fuerza Z + Fuerza
X + Fuerza Y + Fuerza Z]
datocompleto+=datofx;
datocompleto+=datofy;
datocompleto+=datofz;


//Envia la trama completa.
Sleep(500);
Cliente.Send(datocompleto);

            datocompleto="";
            stringlongitudfx="";
    stringlongitudfy="";
            stringlongitudfz="";
            datofx = "0";
            datofy = "0";
    datofz = "0";
            longitudfx=0;
            longitudfy=0;
            longitudfz=0;


}

           // Processor 2

   if (i == 1)
           {
             msg.Format("%d",off.fx);
             m_fx2.SetWindowText(msg);
             m_pfx2.SetPos(abs(off.fx));
    msg.Format("%d",off.fy);
             m_fy2.SetWindowText(msg);
             m_pfy2.SetPos(abs(off.fy));
    msg.Format("%d",off.fz);
             m_fz2.SetWindowText(msg);
```

```
            m_pfz2.SetPos(abs(off.fz));
    msg.Format("%d",off.mx);
            m_mx2.SetWindowText(msg);
            m_pmx2.SetPos(abs(off.mx));

    msg.Format("%d",off.my);
            m_my2.SetWindowText(msg);
            m_pmy2.SetPos(abs(off.my));

            msg.Format("%d",off.mz);
            m_mz2.SetWindowText(msg);
            m_pmz2.SetPos(abs(off.mz));

            //off.v2=(int) sqrt(off.fx*off.fx+off.fy*off.fy+off.fz*off.fz);

    msg.Format("%d",off.v1);
            m_v3.SetWindowText(msg);

            m_pv3.SetPos(abs(off.v1));
    msg.Format("%d",off.v2);
            m_v4.SetWindowText(msg);
            m_pv4.SetPos(abs(off.v2));
           }

          // Processor 3

  if (i == 2)
           {
            msg.Format("%d",off.fx);
            m_fx3.SetWindowText(msg);
            m_pfx3.SetPos(abs(off.fx));
    msg.Format("%d",off.fy);
            m_fy3.SetWindowText(msg);
            m_pfy3.SetPos(abs(off.fy));
    msg.Format("%d",off.fz);
            m_fz3.SetWindowText(msg);
            m_pfz3.SetPos(abs(off.fz));
    msg.Format("%d",off.mx);
            m_mx3.SetWindowText(msg);
            m_pmx3.SetPos(abs(off.mx));

    msg.Format("%d",off.my);
            m_my3.SetWindowText(msg);
            m_pmy3.SetPos(abs(off.my));

            msg.Format("%d",off.mz);
            m_mz3.SetWindowText(msg);
            m_pmz3.SetPos(abs(off.mz));

    msg.Format("%d",off.v1);
            m_v5.SetWindowText(msg);
            m_pv5.SetPos(abs(off.v1));
    msg.Format("%d",off.v2);
            m_v6.SetWindowText(msg);
            m_pv6.SetPos(abs(off.v2));
           }
```

```
            // Processor 4

    if (i == 3)
            {
              msg.Format("%d",off.fx);
              m_fx4.SetWindowText(msg);
              m_pfx4.SetPos(abs(off.fx));
     msg.Format("%d",off.fy);
              m_fy4.SetWindowText(msg);
              m_pfy4.SetPos(abs(off.fy));
     msg.Format("%d",off.fz);
              m_fz4.SetWindowText(msg);
              m_pfz4.SetPos(abs(off.fz));
     msg.Format("%d",off.mx);
              m_mx4.SetWindowText(msg);
              m_pmx4.SetPos(abs(off.mx));

     msg.Format("%d",off.my);
              m_my4.SetWindowText(msg);
              m_pmy4.SetPos(abs(off.my));

              msg.Format("%d",off.mz);
              m_mz4.SetWindowText(msg);
              m_pmz4.SetPos(abs(off.mz));

     msg.Format("%d",off.v1);
              m_v7.SetWindowText(msg);
              m_pv7.SetPos(abs(off.v1));
     msg.Format("%d",off.v2);
              m_v8.SetWindowText(msg);
              m_pv8.SetPos(abs(off.v2));

            }
            }

            CDialog::OnTimer(nIDEvent);

}

void CJr3pciDlg::Oninidown()
{


}

void CJr3pciDlg::OndetFS()
{
   struct six_axis_array fs;
   CString msg;

   m_fsfx.GetWindowText(msg);
          fs.fx=atoi(msg);
   m_fsfy.GetWindowText(msg);
          fs.fy=atoi(msg);
   m_fsfz.GetWindowText(msg);
```

```
        fs.fz=atoi(msg);
   m_fsmx.GetWindowText(msg);
        fs.mx=atoi(msg);
   m_fsmy.GetWindowText(msg);
        fs.my=atoi(msg);
   m_fsmz.GetWindowText(msg);
        fs.mz=atoi(msg);

        m_pnum.GetWindowText(msg);
        pnum=(short)strtoul(msg,NULL,10);

        set_full_scales(fs,pnum);
        m_msg.SetWindowText("New FScales Programmed");


}

void CJr3pciDlg::OnrMfs()
{
   struct six_axis_array fs;
   CString msg;

        m_pnum.GetWindowText(msg);
        pnum=(short)strtoul(msg,NULL,10);

        fs=get_recommended_full_scales(MAX_F_S,pnum);
        msg.Format("%d",fs.fx);
        m_fsfx.SetWindowText(msg);
   msg.Format("%d",fs.fy);
        m_fsfy.SetWindowText(msg);
   msg.Format("%d",fs.fz);
        m_fsfz.SetWindowText(msg);
   msg.Format("%d",fs.mx);
        m_fsmx.SetWindowText(msg);
   msg.Format("%d",fs.my);
        m_fsmy.SetWindowText(msg);
   msg.Format("%d",fs.mz);
        m_fsmz.SetWindowText(msg);
        //CJr3pciDlg::OndetFS();
        m_msg.SetWindowText("FScales Read: MAX_F_S");

}

void CJr3pciDlg::Onrminfs()
{

   struct six_axis_array fs;
   CString msg;

        m_pnum.GetWindowText(msg);
        pnum=(short)strtoul(msg,NULL,10);

        fs=get_recommended_full_scales(MIN_F_S,pnum);
        msg.Format("%d",fs.fx);
        m_fsfx.SetWindowText(msg);
   msg.Format("%d",fs.fy);
```

```
        m_fsfy.SetWindowText(msg);
   msg.Format("%d",fs.fz);
        m_fsfz.SetWindowText(msg);
   msg.Format("%d",fs.mx);
        m_fsmx.SetWindowText(msg);
   msg.Format("%d",fs.my);
        m_fsmy.SetWindowText(msg);
   msg.Format("%d",fs.mz);
        m_fsmz.SetWindowText(msg);
        m_msg.SetWindowText("FScales Read: MIN_F_S");

}

void CJr3pciDlg::Onsetoff()
{
   struct six_axis_array off;
   CString msg;

        m_pnum.GetWindowText(msg);
        pnum=(short)strtoul(msg,NULL,10);

        change_offset_num(atoi(msg),pnum);
        use_offset(atoi(msg),0);


   m_off_fx.GetWindowText(msg);
        off.fx=atoi(msg);
   m_off_fy.GetWindowText(msg);
        off.fy=atoi(msg);
   m_off_fz.GetWindowText(msg);
        off.fz=atoi(msg);
   m_off_mx.GetWindowText(msg);
        off.mx=atoi(msg);
   m_off_my.GetWindowText(msg);
        off.my=atoi(msg);
   m_off_mz.GetWindowText(msg);
        off.mz=atoi(msg);
   m_offnum.GetWindowText(msg);
        set_offsets(off,0);
   m_msg.SetWindowText("New Offsets Programmed");

}

void CJr3pciDlg::Oninfo()
{
        //Variable que habilita el fichero para almacenar los datos leidos por el JR3.
         almacenardatos=1;
         struct six_axis_array off;
         struct force_array fs;
         CString msg;

         m_pnum.GetWindowText(msg);
         pnum=(short)strtoul(msg,NULL,10);

         msg.Format("%d",read_jr3(OFFSET_N,pnum));
         m_offnum.SetWindowText(msg);
```

```
    msg.Format("%d",read_jr3(MODEL_N,pnum));
            m_model.SetWindowText(msg);
    msg.Format("%d",read_jr3(SERIAL_N,pnum));
            m_serial.SetWindowText(msg);
    msg.Format("%d",read_jr3(EEPROM_V_N,pnum));
            m_eeprom.SetWindowText(msg);
    msg.Format("%d",read_jr3(SOFTWARE_V_N,pnum));
            m_software.SetWindowText(msg);
    msg.Format("%d",read_jr3(THICKNESS,pnum));
            m_thickness.SetWindowText(msg);
    msg.Format("%d",read_jr3(UNITS,pnum));
            m_units.SetWindowText(msg);
    msg.Format("%d",read_jr3(BITS,pnum));
            m_adc.SetWindowText(msg);

            off = read_offsets(pnum);
    msg.Format("%d",off.fx);
            m_off_fx.SetWindowText(msg);
    msg.Format("%d",off.fy);
            m_off_fy.SetWindowText(msg);
    msg.Format("%d",off.fz);
            m_off_fz.SetWindowText(msg);
    msg.Format("%d",off.mx);
            m_off_mx.SetWindowText(msg);
    msg.Format("%d",off.my);
            m_off_my.SetWindowText(msg);
    msg.Format("%d",off.mz);
            m_off_mz.SetWindowText(msg);

            fs = get_full_scales(pnum);
    msg.Format("%d",fs.fx);
            m_fsfx.SetWindowText(msg);
    msg.Format("%d",fs.fy);
            m_fsfy.SetWindowText(msg);
    msg.Format("%d",fs.fz);
            m_fsfz.SetWindowText(msg);
    msg.Format("%d",fs.mx);
            m_fsmx.SetWindowText(msg);
    msg.Format("%d",fs.my);
            m_fsmy.SetWindowText(msg);
    msg.Format("%d",fs.mz);
            m_fsmz.SetWindowText(msg);
            m_msg.SetWindowText("Reading INFO from Sensor: Offsets + FScales + Labels");

    msg.Format("%d",read_jr3(VECT_A,pnum));
            m_vect.SetWindowText(msg);


}

BOOL CJr3pciDlg::PreTranslateMessage(MSG* pMsg)
{

        {
```

```
                m_tooltip.RelayEvent(pMsg);
        }
        return CDialog::PreTranslateMessage(pMsg);
}

void CJr3pciDlg::OnDropdownpnumcombo()
{
        short i;
        CString msg;

        m_pnum.ResetContent();
        for (i=0;i<=nump-1;i++)
        {
          msg.Format("%d",i);
          m_pnum.InsertString(i,msg);
        }
}

void CJr3pciDlg::OnSelendokpnumcombo()
{
 short i;
 CString msg;

 i=m_pnum.GetCurSel();
 msg.Format("%d",i);
 m_pnum.SetWindowText(msg);

}

void CJr3pciDlg::OnSelendcancelpnumcombo()
{

}

void CJr3pciDlg::Onabout()
{
  CAboutDlg dlgAbout;
  dlgAbout.DoModal();

}



void ClienteTest::onConnect()
{
        //Variable que indica al conectar que es el primer ciclo de envio
        //de datos para evitar mandar datos erroneos.
   primeravez=1;

 }


void CJr3pciDlg::OnStnClickedmx3()
{

}
```

12.1.4. ProgDlg.cpp

```
//  ProgDlg.cpp : implementation file
// CG: This file was added by the Progress Dialog component

#include "stdafx.h"
#include "resource.h"
#include "ProgDlg.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CProgressDlg dialog

CProgressDlg::CProgressDlg(UINT nCaptionID)
{
        m_nCaptionID = CG_IDS_PROGRESS_CAPTION;
        if (nCaptionID != 0)
                m_nCaptionID = nCaptionID;

   m_bCancel=FALSE;
   m_nLower=0;
   m_nUpper=100;
   m_nStep=1;
   //{{AFX_DATA_INIT(CProgressDlg)
   // NOTE: the ClassWizard will add member initialization here
   //}}AFX_DATA_INIT
   m_bParentDisabled = FALSE;
}

CProgressDlg::~CProgressDlg()
{
   if(m_hWnd!=NULL)
     DestroyWindow();
}

BOOL CProgressDlg::DestroyWindow()
{
   ReEnableParent();
   return CDialog::DestroyWindow();
}

void CProgressDlg::ReEnableParent()
{
   if(m_bParentDisabled && (m_pParentWnd!=NULL))
     m_pParentWnd->EnableWindow(TRUE);
   m_bParentDisabled=FALSE;
}

BOOL CProgressDlg::Create(CWnd *pParent)
{
   // Get the true parent of the dialog
   m_pParentWnd = CWnd::GetSafeOwner(pParent);
```

```
  // m_bParentDisabled is used to re-enable the parent window
  // when the dialog is destroyed. So we don't want to set
  // it to TRUE unless the parent was already enabled.

  if((m_pParentWnd!=NULL) && m_pParentWnd->IsWindowEnabled())
  {
   m_pParentWnd->EnableWindow(FALSE);
   m_bParentDisabled = TRUE;
  }

  if(!CDialog::Create(CProgressDlg::IDD,pParent))
  {
   ReEnableParent();
   return FALSE;
  }

  return TRUE;
}

void CProgressDlg::DoDataExchange(CDataExchange* pDX)
{
  CDialog::DoDataExchange(pDX);
  //{{AFX_DATA_MAP(CProgressDlg)
  DDX_Control(pDX, CG_IDC_PROGDLG_PROGRESS, m_Progress);
  //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CProgressDlg, CDialog)
  //{{AFX_MSG_MAP(CProgressDlg)
  //}}AFX_MSG_MAP
END_MESSAGE_MAP()

void CProgressDlg::SetStatus(LPCTSTR lpszMessage)
{
  ASSERT(m_hWnd); // Don't call this _before_ the dialog has
          // been created. Can be called from OnInitDialog
  CWnd *pWndStatus = GetDlgItem(CG_IDC_PROGDLG_STATUS);

  // Verify that the static text control exists
  ASSERT(pWndStatus!=NULL);
  pWndStatus->SetWindowText(lpszMessage);
}

void CProgressDlg::OnCancel()
{
  m_bCancel=TRUE;
}

void CProgressDlg::SetRange(int nLower,int nUpper)
{
  m_nLower = nLower;
  m_nUpper = nUpper;
  m_Progress.SetRange(nLower,nUpper);
}
```

```
int CProgressDlg::SetPos(int nPos)
{
   PumpMessages();
   int iResult = m_Progress.SetPos(nPos);
   UpdatePercent(nPos);
   return iResult;
}

int CProgressDlg::SetStep(int nStep)
{
   m_nStep = nStep; // Store for later use in calculating percentage
   return m_Progress.SetStep(nStep);
}

int CProgressDlg::OffsetPos(int nPos)
{
   PumpMessages();
   int iResult = m_Progress.OffsetPos(nPos);
   UpdatePercent(iResult+nPos);
   return iResult;
}

int CProgressDlg::StepIt()
{
   PumpMessages();
   int iResult = m_Progress.StepIt();
   UpdatePercent(iResult+m_nStep);
   return iResult;
}

void CProgressDlg::PumpMessages()
{
   // Must call Create() before using the dialog
   ASSERT(m_hWnd!=NULL);

   MSG msg;
   // Handle dialog messages
   while(PeekMessage(&msg, NULL, 0, 0, PM_REMOVE))
   {
    if(!IsDialogMessage(&msg))
    {
      TranslateMessage(&msg);
      DispatchMessage(&msg);
    }
   }
}

BOOL CProgressDlg::CheckCancelButton()
{
   // Process all pending messages
   PumpMessages();

   // Reset m_bCancel to FALSE so that
   // CheckCancelButton returns FALSE until the user
   // clicks Cancel again. This will allow you to call
   // CheckCancelButton and still continue the operation.
```

```
    // If m_bCancel stayed TRUE, then the next call to
    // CheckCancelButton would always return TRUE

    BOOL bResult = m_bCancel;
    m_bCancel = FALSE;

    return bResult;
}

void CProgressDlg::UpdatePercent(int nNewPos)
{
    CWnd *pWndPercent = GetDlgItem(CG_IDC_PROGDLG_PERCENT);
    int nPercent;

    int nDivisor = m_nUpper - m_nLower;
    ASSERT(nDivisor>0);  // m_nLower should be smaller than m_nUpper

    int nDividend = (nNewPos - m_nLower);
    ASSERT(nDividend>=0);   // Current position should be greater than m_nLower

    nPercent = nDividend * 100 / nDivisor;

    // Since the Progress Control wraps, we will wrap the percentage
    // along with it. However, don't reset 100% back to 0%
    if(nPercent!=100)
      nPercent %= 100;

    // Display the percentage
    CString strBuf;
    strBuf.Format(_T("%d%c"),nPercent,_T('%'));

            CString strCur; // get current percentage
    pWndPercent->GetWindowText(strCur);

            if (strCur != strBuf)
                    pWndPercent->SetWindowText(strBuf);
}

/////////////////////////////////////////////////////////////////////////////
// CProgressDlg message handlers

BOOL CProgressDlg::OnInitDialog()
{
    CDialog::OnInitDialog();
    m_Progress.SetRange(m_nLower,m_nUpper);
    m_Progress.SetStep(m_nStep);
    m_Progress.SetPos(m_nLower);

            CString strCaption;
            VERIFY(strCaption.LoadString(m_nCaptionID));
    SetWindowText(strCaption);

    return TRUE;
            }
```

12.1.5. StdAfx.cpp

```
// stdafx.cpp : source file that includes just the standard includes
//    jr3pci.pch will be the pre-compiled header
//    stdafx.obj will contain the pre-compiled type information

#include "stdafx.h"
```

12.2. Cabeceras .h

12.2.1. DlgProxy.h

```
// DlgProxy.h : header file
//

#if !defined(AFX_DLGPROXY_H__243D6A69_8A6D_4B9C_8B68_8DF1ACA527AD__INCLUDED_)
#define AFX_DLGPROXY_H__243D6A69_8A6D_4B9C_8B68_8DF1ACA527AD__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class CJr3pciDlg;

/////////////////////////////////////////////////////////////////////////////
// CJr3pciDlgAutoProxy command target

class CJr3pciDlgAutoProxy : public CCmdTarget
{
	DECLARE_DYNCREATE(CJr3pciDlgAutoProxy)

	CJr3pciDlgAutoProxy();		// protected constructor used by dynamic creation

// Attributes
public:
	CJr3pciDlg* m_pDialog;

// Operations
public:

// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CJr3pciDlgAutoProxy)
	public:
	virtual void OnFinalRelease();
	//}}AFX_VIRTUAL

// Implementation
protected:
	virtual ~CJr3pciDlgAutoProxy();

	// Generated message map functions
	//{{AFX_MSG(CJr3pciDlgAutoProxy)
		// NOTE - the ClassWizard will add and remove member functions here.
	//}}AFX_MSG

	DECLARE_MESSAGE_MAP()
	DECLARE_OLECREATE(CJr3pciDlgAutoProxy)

	// Generated OLE dispatch map functions
	//{{AFX_DISPATCH(CJr3pciDlgAutoProxy)
		// NOTE - the ClassWizard will add and remove member functions here.
	//}}AFX_DISPATCH
	DECLARE_DISPATCH_MAP()
```

```
	DECLARE_INTERFACE_MAP()
};

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

	#endif //
!defined(AFX_DLGPROXY_H__243D6A69_8A6D_4B9C_8B68_8DF1ACA527AD__INCLUDED_)
```

12.2.2. Jr3Pci.h

```
// jr3pci.h : main header file for the JR3PCI application
//

#if !defined(AFX_JR3PCI_H__59C05E9E_D63A_41AB_842C_A5022DB758DE__INCLUDED_)
#define AFX_JR3PCI_H__59C05E9E_D63A_41AB_842C_A5022DB758DE__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#ifndef __AFXWIN_H__
	#error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"		// main symbols

/////////////////////////////////////////////////////////////////////////////
// CJr3pciApp:
// See jr3pci.cpp for the implementation of this class
//

class CJr3pciApp : public CWinApp
{
public:
	CJr3pciApp();

// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CJr3pciApp)
	public:
	virtual BOOL InitInstance();
	//}}AFX_VIRTUAL

// Implementation

	//{{AFX_MSG(CJr3pciApp)
		// NOTE - the ClassWizard will add and remove member functions here.
		//    DO NOT EDIT what you see in these blocks of generated code !
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};


/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_JR3PCI_H__59C05E9E_D63A_41AB_842C_A5022DB758DE__INCLUDED_)
```

12.2.3. Jr3pciDLg.h

```
// jr3pciDlg.h : header file
#include <SolarSockets/solarSockets++.h>

#if !defined(AFX_JR3PCIDLG_H__78BD5C12_9199_45A1_93AB_356DD59654C2__INCLUDED_)
#define AFX_JR3PCIDLG_H__78BD5C12_9199_45A1_93AB_356DD59654C2__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class CJr3pciDlgAutoProxy;
class ssPPClient;
/////////////////////////////////////////////////////////////////////////////
// CJr3pciDlg dialog

class CJr3pciDlg : public CDialog
{
	DECLARE_DYNAMIC(CJr3pciDlg);
	friend class CJr3pciDlgAutoProxy;
  friend class ssPPClient;

// Construction
public:
	virtual BOOL PreTranslateMessage(MSG* pMsg);
	CJr3pciDlg(CWnd* pParent = NULL);	// standard constructor
	virtual ~CJr3pciDlg();

// Dialog Data
	//{{AFX_DATA(CJr3pciDlg)
	enum { IDD = IDD_JR3PCI_DIALOG };
	CEdit	m_bnum;
	CEdit	m_vect;
	CButtonm_about;
	CComboBox	m_pnum;
	CStatic	m_mz4;
	CStatic	m_mz3;
	CStatic	m_v8;
	CStatic	m_v7;
	CStatic	m_v6;
	CStatic	m_v5;
	CStatic	m_v4;
	CStatic	m_v3;
	CProgressCtrl	m_pv8;
	CProgressCtrl	m_pv7;
	CProgressCtrl	m_pv6;
	CProgressCtrl	m_pv5;
	CProgressCtrl	m_pv4;
	CProgressCtrl	m_pv3;
	CProgressCtrl	m_pmz4;
	CProgressCtrl	m_pmz3;
	CProgressCtrl	m_pmz2;
	CProgressCtrl	m_pmy4;
	CProgressCtrl	m_pmy3;
	CProgressCtrl	m_pmy2;
```

```
CProgressCtrl    m_pmx4;
CProgressCtrl    m_pmx3;
CProgressCtrl    m_pmx2;
CProgressCtrl    m_pfz4;
CProgressCtrl    m_pfz3;
CProgressCtrl    m_pfz2;
CProgressCtrl    m_pfy4;
CProgressCtrl    m_pfy3;
CProgressCtrl    m_pfy2;
CProgressCtrl    m_pfx4;
CProgressCtrl    m_pfx3;
CProgressCtrl    m_pfx2;
CStatic  m_mz2;
CStatic  m_my4;
CStatic  m_my3;
CStatic  m_my2;
CStatic  m_mx4;
CStatic  m_mx3;
CStatic  m_mx2;
CStatic  m_fz4;
CStatic  m_fz3;
CStatic  m_fz2;
CStatic  m_fy4;
CStatic  m_fy3;
CStatic  m_fy2;
CStatic  m_fx4;
CStatic  m_fx3;
CStatic  m_fx2;
CEdit    m_nump;
CEdit    m_did;
CEdit    m_vid;
CEdit    m_offnum;
CStatic  m_msg;
CButtonm_setoff;
CButtonm_rminfs;
CButtonm_rmfs;
CButtonm_info;
CButtonm_setfs;
CButtonm_ok;
CButtonm_inidown;
CButtonm_radio1;
CButtonm_read;
CButtonm_stop;
CStatic  m_v2;
CStatic  m_v1;
CEdit    m_units;
CEdit    m_thickness;
CEdit    m_software;
CEdit    m_serial;
CButtonm_roff;
CProgressCtrl    m_pv2;
CProgressCtrl    m_pv1;
CProgressCtrl    m_pmz;
CProgressCtrl    m_pmy;
CProgressCtrl    m_pmx;
CProgressCtrl    m_pfz;
```

```
        CProgressCtrl    m_pfy;
        CProgressCtrl    m_pfx;
        CEdit    m_off_mz;
        CEdit    m_off_my;
        CEdit    m_off_mx;
        CEdit    m_off_fz;
        CEdit    m_off_fy;
        CEdit    m_off_fx;
        CStatic  m_mz;
        CStatic  m_my;
        CStatic  m_mx;
        CEdit    m_model;
        CButtonm_init;
        CStatic  m_fz;
        CStatic  m_fy;
        CStatic  m_fx;
        CEdit    m_fsmz;
        CEdit    m_fsmy;
        CEdit    m_fsmx;
        CEdit    m_fsfz;
        CEdit    m_fsfy;
        CEdit    m_fsfx;
        CEdit    m_eeprom;
        CEdit    m_adc;
        //}}AFX_DATA

        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CJr3pciDlg)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);        // DDX/DDV support
        //}}AFX_VIRTUAL

// Implementation
protected:
        CToolTipCtrl m_tooltip;
        CJr3pciDlgAutoProxy* m_pAutoProxy;
        HICON m_hIcon;
        HICON m_hIcon1;

        BOOL CanExit();

        // Generated message map functions
        //{{AFX_MSG(CJr3pciDlg)
        virtual BOOL OnInitDialog();
        afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
        afx_msg void OnPaint();
        afx_msg HCURSOR OnQueryDragIcon();
        afx_msg void OnClose();
        virtual void OnOK();
        virtual void OnCancel();
        afx_msg void Oninit();
        afx_msg void Onroff();
        afx_msg void Onread();
        afx_msg void Onstop();
        afx_msg void OnTimer(UINT nIDEvent);
```

```
        afx_msg void Oninidown();
        afx_msg void OndetFS();
        afx_msg void OnrMfs();
        afx_msg void Onrminfs();
        afx_msg void Onsetoff();
        afx_msg void Oninfo();
        afx_msg void OnDropdownpnumcombo();
        afx_msg void OnSelendokpnumcombo();
        afx_msg void OnSelendcancelpnumcombo();
        afx_msg void Onabout();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
public:
        afx_msg void OnStnClickedmx3();
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.


//Declaración de las clase necesaria para la conexión Socket (TCP/IP)
class ClienteTest : public  ssPPClient
{
public:
        ClienteTest(void);
        virtual ~ClienteTest(void);

void onConnect();

};


        #endif                                                                  //
!defined(AFX_JR3PCIDLG_H__78BD5C12_9199_45A1_93AB_356DD59654C2__INCLUDED_)
```

12.2.4. ProgDlg.h

```
/ ProgDlg.h : header file
// CG: This file was added by the Progress Dialog component

/////////////////////////////////////////////////////////////////////////////
// CProgressDlg dialog

#ifndef __PROGDLG_H__
#define __PROGDLG_H__

class CProgressDlg : public CDialog
{
// Construction / Destruction
public:
   CProgressDlg(UINT nCaptionID = 0);   // standard constructor
   ~CProgressDlg();

   BOOL Create(CWnd *pParent=NULL);

   // Checking for Cancel button
   BOOL CheckCancelButton();
   // Progress Dialog manipulation
   void SetStatus(LPCTSTR lpszMessage);
   void SetRange(int nLower,int nUpper);
   int  SetStep(int nStep);
   int  SetPos(int nPos);
   int  OffsetPos(int nPos);
   int  StepIt();

// Dialog Data
   //{{AFX_DATA(CProgressDlg)
   enum { IDD = CG_IDD_PROGRESS };
   CProgressCtrl  m_Progress;
   //}}AFX_DATA

// Overrides
   // ClassWizard generated virtual function overrides
   //{{AFX_VIRTUAL(CProgressDlg)
   public:
   virtual BOOL DestroyWindow();
   protected:
   virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
   //}}AFX_VIRTUAL

// Implementation
protected:
            UINT m_nCaptionID;
   int m_nLower;
   int m_nUpper;
   int m_nStep;

   BOOL m_bCancel;
   BOOL m_bParentDisabled;

   void ReEnableParent();
```

```
    virtual void OnCancel();
    virtual void OnOK() {};
    void UpdatePercent(int nCurrent);
    void PumpMessages();

    // Generated message map functions
    //{{AFX_MSG(CProgressDlg)
    virtual BOOL OnInitDialog();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

#endif // __PROGDLG_H__
```

12.2.5. Resource.h

```
//{{NO_DEPENDENCIES}}
// Microsoft Visual C++ generated include file.
// Used by jr3pci.rc
//
#define IDM_ABOUTBOX                    0x0010
#define IDD_ABOUTBOX                    100
#define IDP_OLE_INIT_FAILED             100
#define IDS_ABOUTBOX                    101
#define IDD_JR3PCI_DIALOG               102
#define CG_IDD_PROGRESS                 103
#define CG_IDS_PROGRESS_CAPTION         104
#define IDR_MAINFRAME                   128
#define IDR_jr3pci                      128
#define IDB_B1                          129
#define IDI_read                        131
#define IDI_ICON3                       132
#define IDI_ICON4                       133
#define IDI_ok                          134
#define IDI_nok                         135
#define IDI_init2                       136
#define IDI_ICON8                       137
#define IDI_ICON9                       138
#define IDI_info                        139
#define IDI_stop                        140
#define IDI_ICON12                      141
#define IDI_ICON13                      142
#define IDI_init                        143
#define IDI_ICON15                      144
#define IDI_jr3                         145
#define IDI_start                       146
#define IDI_out                         148
#define IDI_ICON2                       149
#define IDI_ICON1                       150
#define IDB_PNG1                        158
#define IDB_PNG2                        159
#define IDB_PNG3                        161
#define IDB_PNG4                        166
#define IDB_PNG5                        168
#define IDI_ICON5                       172
#define CG_IDC_PROGDLG_PROGRESS         1003
#define CG_IDC_PROGDLG_PERCENT          1004
#define IDC_init                        1004
#define CG_IDC_PROGDLG_STATUS           1005
#define IDC_fx                          1005
#define IDC_fy                          1006
#define IDC_fz                          1007
#define IDC_v1                          1008
#define IDC_mx                          1009
#define IDC_my                          1010
#define IDC_mz                          1011
#define IDC_v2                          1012
#define IDC_PROGRESS_fx                 1013
#define IDC_roff                        1014
#define IDC_serial                      1015
#define IDC_model                       1016
#define IDC_eeprom                      1017
#define IDC_read                        1018
#define IDC_stop                        1019
#define IDC_PROGRESS_fx2                1020
#define IDC_fx2                         1021
#define IDC_PROGRESS_fy                 1022
```

```
#define IDC_PROGRESS_fz                 1023
#define IDC_PROGRESS_v1                 1024
#define IDC_PROGRESS_mx                 1025
#define IDC_PROGRESS_my                 1026
#define IDC_PROGRESS_mz                 1027
#define IDC_PROGRESS_v2                 1028
#define IDC_setoff                      1029
#define IDC_software                    1030
#define IDC_detFS                       1031
#define IDC_thickness                   1032
#define IDC_units                       1033
#define IDC_adc                         1034
#define IDC_off_fx                      1035
#define IDC_off_fy                      1036
#define IDC_off_fz                      1037
#define IDC_off_mx                      1038
#define IDC_off_my                      1039
#define IDC_off_mz                      1040
#define IDC_fs_fx                       1041
#define IDC_fs_fy                       1042
#define IDC_fs_fz                       1043
#define IDC_fs_mx                       1044
#define IDC_fs_my                       1045
#define IDC_fs_mz                       1046
#define IDC_rMfs                        1047
#define IDC_rminfs                      1048
#define IDC_msg                         1049
#define IDC_info                        1050
#define IDC_offnum                      1051
#define IDC_vid                         1052
#define IDC_did                         1053
#define IDC_nump                        1054
#define IDC_vect                        1055
#define IDC_PROGRESS_fy2                1056
#define IDC_fy2                         1057
#define IDC_PROGRESS_fz2                1058
#define IDC_fz2                         1059
#define IDC_PROGRESS_v3                 1060
#define IDC_v3                          1061
#define IDC_PROGRESS_mx2                1062
#define IDC_mx2                         1063
#define IDC_PROGRESS_my2                1064
#define IDC_my2                         1065
#define IDC_PROGRESS_mz2                1066
#define IDC_mz2                         1067
#define IDC_PROGRESS_v4                 1068
#define IDC_v4                          1069
#define IDC_pnumcombo                   1070
#define IDC_bnum                        1071
#define IDC_PROGRESS_fx5                1072
#define IDC_fx5                         1073
#define IDC_PROGRESS_fy5                1074
#define IDC_fy5                         1075
#define IDC_PROGRESS_fz5                1076
#define IDC_fz5                         1077
#define IDC_PROGRESS_v9                 1078
#define IDC_v9                          1079
#define IDC_PROGRESS_mx5                1080
#define IDC_mx5                         1081
#define IDC_PROGRESS_my5                1082
#define IDC_my5                         1083
#define IDC_PROGRESS_mz5                1084
#define IDC_mz5                         1085
```

```
#define IDC_PROGRESS_v10                1086
#define IDC_v10                         1087
#define IDC_vect2                       1088
#define IDC_PROGRESS_fx3                1102
#define IDC_fx3                         1103
#define IDC_PROGRESS_fy3                1104
#define IDC_fy3                         1105
#define IDC_PROGRESS_fz3                1106
#define IDC_fz3                         1107
#define IDC_PROGRESS_v5                 1108
#define IDC_v5                          1109
#define IDC_PROGRESS_mx3                1110
#define IDC_mx3                         1111
#define IDC_PROGRESS_my3                1112
#define IDC_my3                         1113
#define IDC_PROGRESS_mz3                1114
#define IDC_mz3                         1115
#define IDC_PROGRESS_v6                 1116
#define IDC_v6                          1117
#define IDC_PROGRESS_fx4                1118
#define IDC_fx4                         1119
#define IDC_PROGRESS_fy4                1120
#define IDC_fy4                         1121
#define IDC_PROGRESS_fz4                1122
#define IDC_fz4                         1123
#define IDC_PROGRESS_v7                 1124
#define IDC_v7                          1125
#define IDC_PROGRESS_mx4                1126
#define IDC_mx4                         1127
#define IDC_PROGRESS_my4                1128
#define IDC_my4                         1129
#define IDC_PROGRESS_mz4                1130
#define IDC_mz4                         1131
#define IDC_PROGRESS_v8                 1132
#define IDC_v8                          1133
#define IDC_about                       1134
#define IDC_PROGRESS_fx6                1135
#define IDC_fx6                         1136
#define IDC_PROGRESS_fy6                1137
#define IDC_fy6                         1138
#define IDC_PROGRESS_fz6                1139
#define IDC_fz6                         1140
#define IDC_PROGRESS_v11                1141
#define IDC_v11                         1142
#define IDC_PROGRESS_mx6                1143
#define IDC_mx6                         1144
#define IDC_PROGRESS_my6                1145
#define IDC_my6                         1146
#define IDC_PROGRESS_mz6                1147
#define IDC_mz6                         1148
#define IDC_PROGRESS_v12                1149
#define IDC_v12                         1150
#define IDC_PROGRESS_fx7                1151
#define IDC_fx7                         1152
#define IDC_PROGRESS_fy7                1153
#define IDC_fy7                         1154
#define IDC_PROGRESS_fz7                1155
#define IDC_fz7                         1156
#define IDC_PROGRESS_v13                1157
#define IDC_v13                         1158
#define IDC_PROGRESS_mx7                1159
#define IDC_PROGRESS_my7                1160
#define IDC_PROGRESS_mz7                1161
```

```
#define IDC_PROGRESS_v14                1162
#define IDC_about2                      1163

// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_NEXT_RESOURCE_VALUE        173
#define _APS_NEXT_COMMAND_VALUE         32771
#define _APS_NEXT_CONTROL_VALUE         1072
#define _APS_NEXT_SYMED_VALUE           105
#endif
      #endif
```

12.2.6. StdAfx.h

```
// stdafx.h : include file for standard system include files,
//  or project specific include files that are used frequently, but
//      are changed infrequently
//

#if !defined(AFX_STDAFX_H__01EB288E_B723_4249_AFAD_220A79F6A34E__INCLUDED_)
#define AFX_STDAFX_H__01EB288E_B723_4249_AFAD_220A79F6A34E__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#define VC_EXTRALEAN            // Exclude rarely-used stuff from Windows headers

#include <afxwin.h>         // MFC core and standard components
#include <afxext.h>         // MFC extensions
#include <afxdisp.h>        // MFC Automation classes
//#include <afxdtctl.h>             // MFC support for Internet Explorer 4 Common Controls
#ifndef _AFX_NO_AFXCMN_SUPPORT
#include <afxcmn.h>                     // MFC support for Windows Common Controls
#endif // _AFX_NO_AFXCMN_SUPPORT


// This macro is the same as IMPLEMENT_OLECREATE, except it passes TRUE
//  for the bMultiInstance parameter to the COleObjectFactory constructor.
//  We want a separate instance of this application to be launched for
//  each automation proxy object requested by automation controllers.
#ifndef IMPLEMENT_OLECREATE2
#define IMPLEMENT_OLECREATE2(class_name, external_name, l, w1, w2, b1, b2, b3, b4, b5, b6,
b7, b8) \
        AFX_DATADEF COleObjectFactory class_name::factory(class_name::guid, \
                RUNTIME_CLASS(class_name), TRUE, _T(external_name)); \
        const AFX_DATADEF GUID class_name::guid = \
                { l, w1, w2, { b1, b2, b3, b4, b5, b6, b7, b8 } };
#endif // IMPLEMENT_OLECREATE2

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

        #endif                                                                          //
!defined(AFX_STDAFX_H__01EB288E_B723_4249_AFAD_220A79F6A34E__INCLUDED_)
```

12.3. Archivo de RAPID.

```
MODULE MainModule
!Declaración de variables para creación del socket
VAR socketdev temp_socket;
VAR socketdev client_socket;
VAR string received_string;
VAR bool keep_listening := TRUE;
VAR NUM conectar:=0;
!Variables de fuerza recibidas.
VAR NUM numfx:=0;
VAR NUM numfy:=0;
VAR NUM numfz:=0;
VAR NUM offfx:=0;
VAR NUM offfy:=0;
VAR NUM offfz:=0;
VAR bool bOK;
VAR num lenfx:=0;
VAR num lenfy:=0;
VAR num lenfz:=0;
VAR robtarget posactual;



!Proceso leer datos, se ejecuta en paralelo con el movimiento.
PROC leer()
!Recepción del dato enviado por el PC.
SocketReceive client_socket \Str:=received_string;
!Extracción de las longitudes de los valores de fuerza.
bOK:=StrToVal (StrPart(received_string,1,1),lenfx);
bOK:=StrToVal (StrPart(received_string,2,1),lenfy);
bOK:=StrToVal (StrPart(received_string,3,1),lenfz);
!Extracción de los valores de fuerza.
bOK:=StrToVal (StrPart(received_string,4,lenfx),numfx);
bOK:=StrToVal (StrPart(received_string,4+lenfx,lenfy),numfy);
bOK:=StrToVal
(StrPart(received_string,4+lenfx+lenfy,lenfz),numfz);
!Valor nulo al string.
received_string := "";
!División de los valores de fuerza para realizar movimientos mas
cortos.
offfx:=numfx/(-100);
offfy:=numfy/(-100);
offfz:=numfz/(-100);


ENDPROC

PROC main()

!Creación del socket y espera de conexión. Solo se realiza la
primera vez.
IF (conectar=0) THEN
SocketCreate temp_socket;
SocketBind temp_socket, "192.168.133.2", 4410;
SocketListen temp_socket;
SocketAccept temp_socket, client_socket;
!Desactivación de la supervisión del movimiento.
MotionSup \Off;
WaitTime 5;
conectar:=1;
```

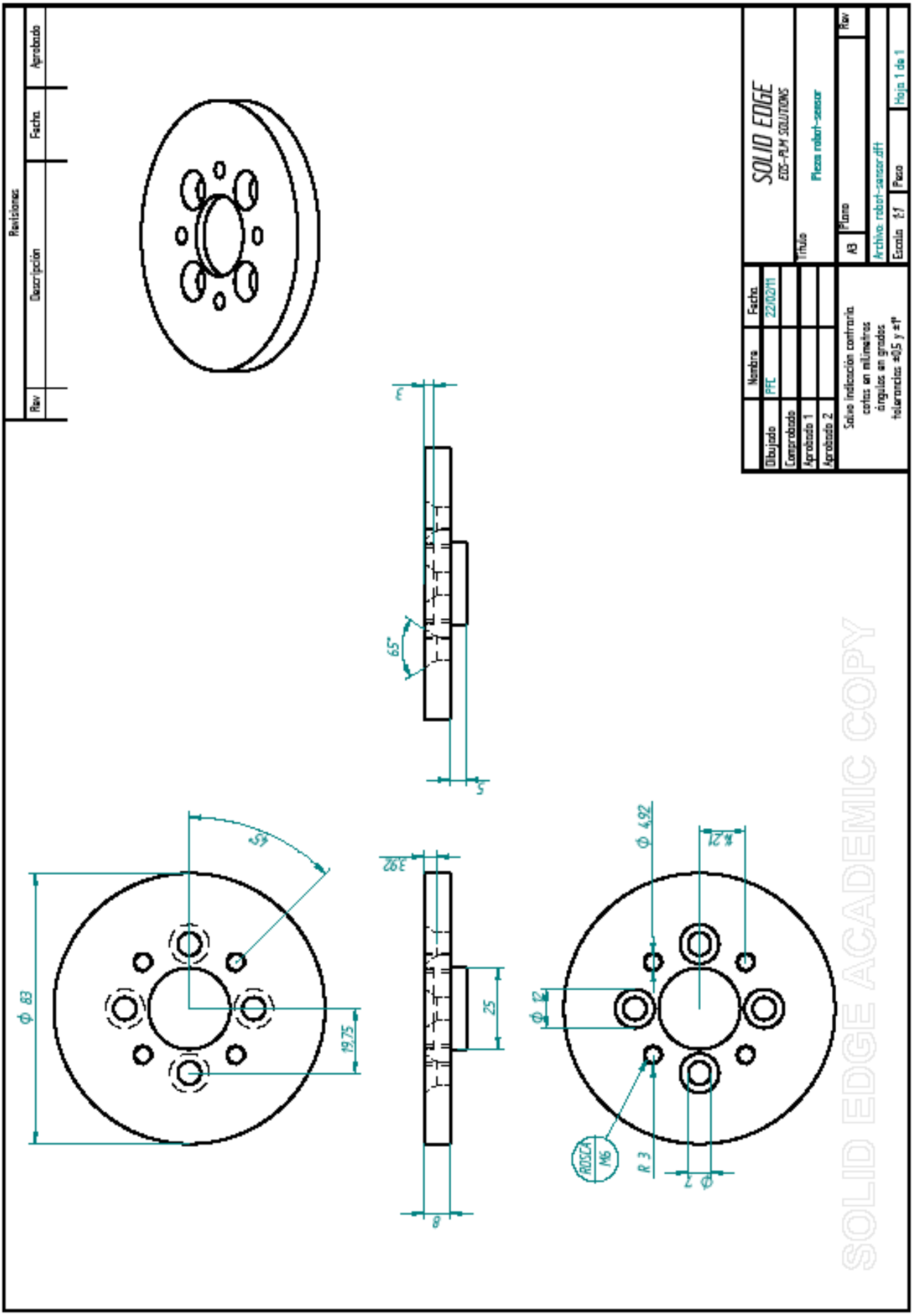
```
      ENDIF
      !Leer la posición actual.
      posactual := CRobT(\Tool:=tool0 \WObj:=wobj0);
      !Movimiento con el offset recibido y mientras ejecutar el
proceso de leer datos.
      MoveJSync Offs (posactual,offfx,offfy,offfz), v100, z50, tool0,
"leer";

      !Condición para desconectar. En este caso el robot permanece
conectado siempre al PC.
      IF (conectar=3) THEN
      SocketClose client_socket;
      SocketClose temp_socket;
      ENDIF

      ENDPROC

      ENDMODULE
```

12.4. Plano pieza de unión robot sensor.



12.5. Plano pieza unión sensor soporte.